uc3m | Universidad Carlos III de Madrid

Ingeniería en Tecnologías Industriales
2017-2018

Trabajo Fin de Grado

Perfeccionamiento de un simulador de incendios forestales basado en Fast Marching

Ricardo González Pacheco
Dr. Santiago Garrido Bullón
10 octubre, 2018

AGRADECIMIENTOS

A mis padres por su sacrificio, admiración, confianza y cuidado continuo.

A mi hermana por su apoyo, cariño y alegría.

A mis abuelos por su aportación emocional y material para conseguir superar este TFG y el grado.

A mi tutor, Santiago, que desde el primer momento confió en mí para continuar su proyecto, por su ayuda y flexibilidad horaria, por su entusiasmo y por su trato.

A mi buen amigo Miquel, por su implicación y amistad tanto en Madrid como desde la distancia.

RESUMEN

Debido a su situación geográfica, España contiene numerosas zonas áridas en las que los incendios forestales no sólo son habituales sino también devastadores e incontrolables en determinadas ocasiones. A pesar de la existencia de importante industria y normativa dedicada a la extinción de éstos, los modelos de previsión en esta área tienen potencial de mejora.

Esta es la razón del alumno de aprovechar su Trabajo de Fin de Grado para contribuir en la creación de una herramienta analítica que dote al país de capacidad predictiva cuando aparezca un incendio. Para ello Fast Marching, ampliamente utilizado en el mundo de la robótica para definir trayectorias, es el algoritmo entorno al cual se construye el programa de este proyecto, un simulador de incendios forestales.

El simulador lo ha iniciado otro alumno y permite al usuario visualizar la evolución de un incendio en 3D en una porción de mapa real a seleccionar, normalmente zona montañosa. Donde previamente se han introducido *inputs* como el punto inicial y final de la simulación, la dirección y velocidad del viento y tiempo máximo de estudio. Además, cuenta con funcionalidades extra como observar el terreno en 3D antes de producirse el incendio y el mapa de combustible de la zona seleccionada.

Sin embargo, no tiene en cuenta una característica básica del viento, factor clave en la propagación de un incendio forestal, como es la variabilidad de la dirección del viento en función de la orografía. Para ejemplificar esta situación basta con observar que el viento avanza con menor velocidad al encontrarse un obstáculo (montaña) que al atravesar una llanura. Dicha optimización es la que se llevará a cabo durante este proyecto, que resulta ser visiblemente necesaria en aras de convertir esta propuesta universitaria en un recurso real para los profesionales que combaten estos desastres.

La tecnología elegida para desarrollar esta mejora es la misma con la que nació el programa, MatLab, pues se trata de la herramienta idónea debido a su potencia matemática y facilidad para trabajar con matrices de grandes dimensiones.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1

INTRODUCCIÓN

1.1. Motivación

Debido al desarrollo de la industria moderna y de la sociedad consumista actual, el planeta Tierra ha sufrido durante las últimas décadas relevantes cambios medioambientales como el efecto invernadero, el deterioro de la capa de ozono, el deshielo de los glaciares o la lluvia ácida. Todos ellos perjudiciales para la supervivencia del ser humano. Algunos afectan a las condiciones climatológicas provocando que las temperaturas cada vez se vuelven más extremas. En verano más calor. En invierno más frío.

Evidentemente está situación viene acompañada por el aumento de catástrofes naturales. Terremotos, tormentas, huracanes, sequías, aumento del nivel del mar. Y por supuesto, la parte que ocupa este trabajo, incendios. Con temperaturas tan dispares, ciertos lares secos aumentan seriamente sus probabilidades de sufrir vastos incendios durante la época estival. Por tanto, tiene sentido que en España se sufra tanto esta catástrofe en concreto, sobre todo durante los meses más calurosos.

En este sentido, la *Agencia Europea de Medio Ambiente* ha publicado un informe [1] en el que anima a los países a tomar acción en los siguientes términos: "El grado de devastación de incendios forestales, inundaciones y tormentas ha demostrado que el coste de no actuar sobre el cambio climático es extremadamente alto". Además, especifica la situación futura para el sur europeo: "Las sequías aumentarán en frecuencia, duración y gravedad en la mayor parte del continente, con un aumento más fuerte en los países del sur donde se espera que, en un clima más cálido, las temporadas de incendios sean más largas", entre los que se encuentra España.

Cabe destacar que el año pasado fue uno de los años más dañados por incendios en la historia de la península. Como afirma *WWF* [2], "En 2017 más de medio millón de hectáreas ardieron, y lo que es peor, perecieron más de un centenar de personas en España y Portugal, por lo que los incendios han pasado de ser un problema forestal a convertirse en una emergencia social". Adicionalmente, en su informe anual [3], declara que los incendios forestales grandes (GIF – mayores de 500 hectáreas) aumentaron casi un 200 por ciento en relación a la media de la última década en España, sobre todo, en la zona noroeste. Nuestro país vecino Portugal, no es ajeno a esta problemática pues, según información extraída de este informe se quemaron casi 500000 hectáreas, cantidad que cuadruplica a la media de la última década dentro de sus fronteras.

Con el objetivo de ofrecer al lector una perspectiva general sobre la situación de la sociedad en relación a los incendios forestales, se explican brevemente los siguientes puntos. Realidad e impacto de los incendios forestales en términos numéricos. Casos reales actuales en España. Causas principales que los originan. Soluciones implementadas a nivel transversal.

1.1.1. Datos numéricos sobre incendios en España

Analizando datos recogidos en el avance informativo 1 enero – 31 diciembre 2017 perteneciente al Área de Defensa contra Incendios Forestales (ADCIF) del Ministerio de Agricultura y Pesca, Alimentación y Medio Ambiente (MAPAMA), se estudia nacionalmente este tipo de catástrofe natural.

En la figura 1.1 se visualizan la evolución de los siniestros y las superficies quemadas (arboladas y desarboladas) a causa de incendios en el periodo temporal desde 1960, que ya contaba con la existencia del Servicio de Incendios Forestales cuya función era producir estadísticas sobre incendios, hasta 2010.

Destaca en el gráfico la reducción considerable de superficie desarbolada (también no arbolada) quemada que se produce a partir del año 1991. Previamente, hasta en 4 años distintos se llegaron a su superar las 400000 hectáreas quemadas de superficie despoblada (2% de la superficie de España). La causa principal de esta reducción es mayoritariamente el desarrollo y perfeccionamiento de los servicios y métodos de extinción, así como las herramientas para llevarlos a cabo. Gracias a esto se consigue reducir la superficie no arbolada quemada total a casi un cuarto, es decir, 100000 hectáreas. Sin embargo, no pasa desapercibido que el número de siniestros aumenta de manera regular hasta 2005, donde el aumento de concienciación y castigo para infractores reduce esta cifra casi a la mitad.

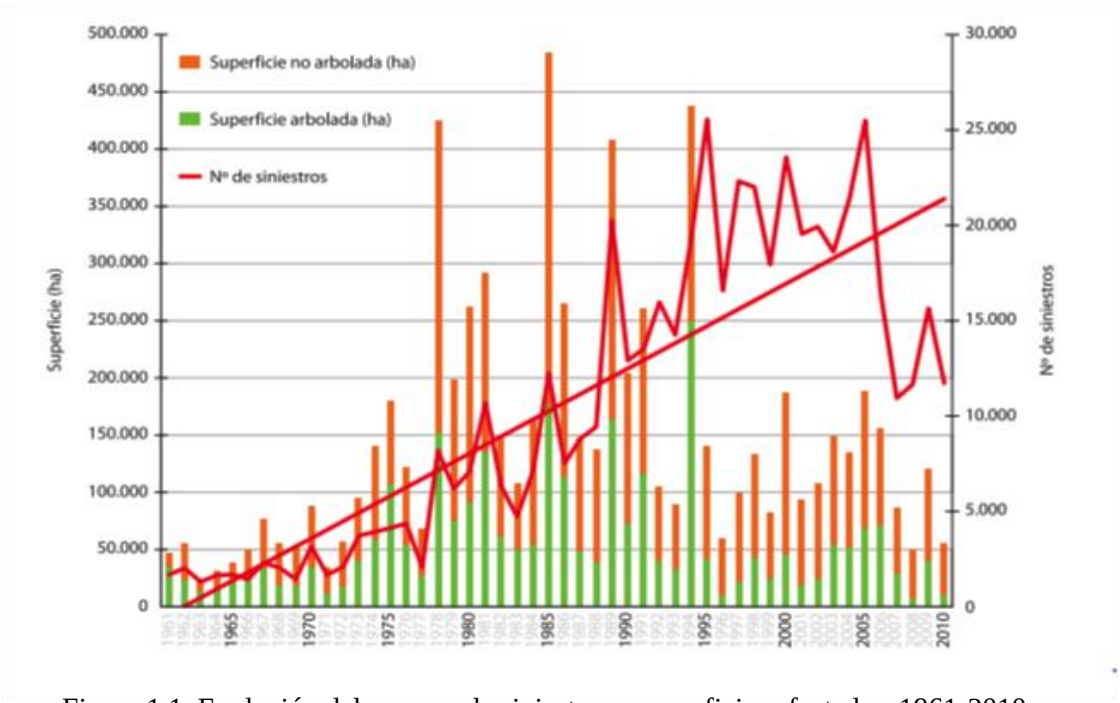


Figura 1.1. Evolución del numero de siniestros y superficies afectadas, 1961-2010

Previamente se ha comentado el dramático 2017 que España ha sufrido en términos de grandes incendios y daños ocasionados. La figura 1.2. refuta esta realidad en

términos absolutos numéricos, hace alusión a los incendios y superficies que quemaron comparando la media de la última década con los ocurridos en 2017. Queda claramente reflejado el alcance del 2017, supera a la década anterior en el número de conatos e incendios. Sin embargo, el dato que realmente ratifica dicho impacto es la superficie quemada. Casi triplica al periodo del decenio anterior en el caso de extensión arbolada (66839 frente a 27226 hectáreas) y casi duplica también al decenio en zonas forestales (178234 frente a 91847).

		MEDIA DEL DECENIO 2007-2017	AÑO 2017
SINIESTROS	NÚMERO DE CONATOS (1<ha)	8228	8705
	NÚMERO DE INCENDIOS (1≥ha)	4135	5088
	TOTAL INCENDIOS	12363	13793
SUPERFICIE QUEMADA	SUPERFICIE ARBOLADA (ha)	27226	66839
	SUPERFICIE FORESTAL (ha)	91847	178234

Figura 1.2. Superficie quemada y siniestros de 2017 frente al decenio 2007 – 2017

1.1.2. Orígenes de los incendios forestales

En la figura 1.3. se visualizan las causas predominantes de aparición de incendios. Los más destacados en España son:

- Causas involuntarias: Se trata de negligencias humanas. Es el caso de incendios que se producen tras quemar pastos, hacer una fogata o tirar una colilla al suelo.

- Causas meteorológicas: Las tormentas eléctricas ejemplifican este grupo. Si un trueno impacta en un material seco como madera se puede producir uno instantáneamente.

- Hijos de fuegos anteriores: Determinadas zonas quemadas que no han sido correctamente apagadas pueden renacer con el viento como principal culpable para volver a formarse un incendio.

- Acciones premeditadas: Se refiere a todo aquel movimiento humano que sea susceptible de generar un fuego llevado a cabo aun siendo conocedor del riesgo que éste implica.

La figura 1.3. profundiza en las distintas vertientes en las que se dividen las acciones premeditadas. Destaca la quema de material agrícola, de pastos para su rehabilitación y de pirómanos (personas con trastorno psicológico a las que les produce placer quemar objetos).

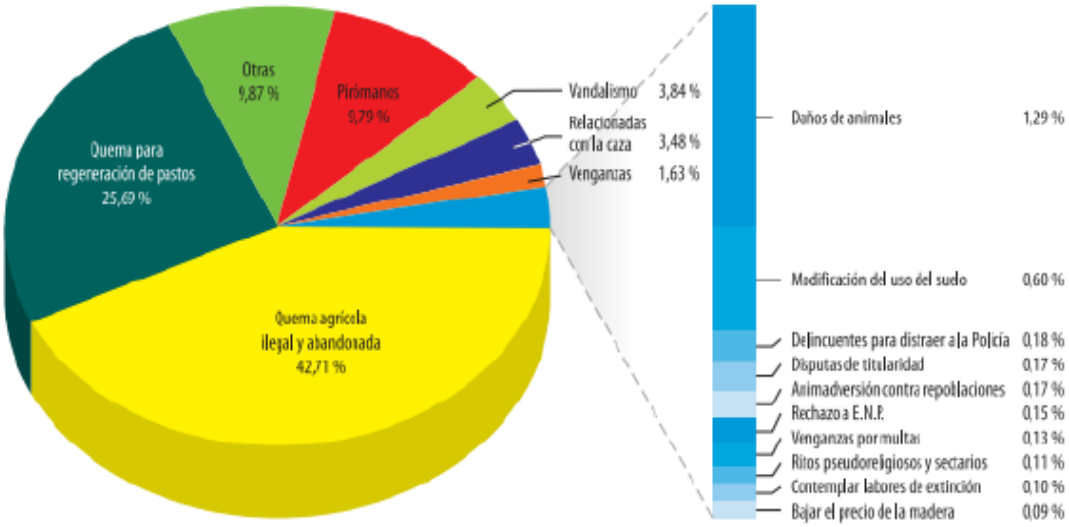


Figura 1.3. División de las principales acciones premeditadas

1.1.3. Pérdidas humanas

Toda persona que experimenta en primera persona un incendio destaca la incertidumbre del desenlace y del rumbo de las llamas, junto con el espanto que genera visualmente y la velocidad que puede alcanzar si el lugar de los hechos es un terreno natural como una montaña.

No es de extrañar pues, que se produzcan tantas muertes en el proceso de extinción de éstos, así como heridos de carácter irreparable con quemaduras graves.

En relación a víctimas mortales, los bomberos desgraciadamente quedan acorralados por las llamas y conforman uno de los grupos más importantes de fallecidos. Los acompañan los accidentes de cualquier índole debido a la escasa visibilidad que produce el humo de las llamas, tanto aéreos como marítimos o terrestres. Por último, se producen bajas humanas durante las evacuaciones donde los civiles están afectados y en el deseo de sobrevivir cometen errores fatales. Un matiz que considerar es la imposibilidad para determinar el camino que seguirá el fuego y como variará con los elementos del entorno como viento u orografía. Es en este punto donde asoma la necesidad de programas como el que se explica en este proyecto de investigación donde predecir el camino crítico del incendio es la principal finalidad.

Entre los principales daños no mortales que pueden causar los incendios forestales se encuentran las quemaduras, enfermedades respiratorias irreversibles provocadas por el humo o aparición de cáncer.

La figura 1.4. resume la cantidad de afectados en la primera década del sigo XXI por incendios forestales.

Año	Nº de personas fallecidas						Nº heridos entre el personal de extinción	Ajenos al personal de extinción
	Personal en labores de extinción o en labores de ida o vuelta de los incendios					Ciudadanos ajenos al personal de extinción		
	Tripulaciones	Agentes forestales y Brigadistas	Personal de Maquin./ Autobombas	Bomberos	Voluntarios			
2001	3	1	0	1	0	0	56	1
2002	3	0	1	0	0	1	60	0
2003	2	1	2	3	0	7	90	2
2004	1	2	0	0	0	2	46	0
2005	4	13	0	0	0	0	101	1
2006	0	0	1	0	0	0	58	0
2007	0	1	0	0	0	0	41	0
2008	0	0	0	0	0	1	27	0
2009	0	1	0	6	0	1	64	0
2010	2	3	0	0	0	5	11	8
TOTALES	15	22	4	10	0	17	554	12

Figura 1.4. Resumen de daños humanos en el primer decenio del sigo XXI

1.1.4. Casos destacados

Por encima de la mayoría de los incendios, un grupo reducido de ellos ha destacado sobre los demás por su gravedad e impacto en la sociedad. En la figura 1.3. se presenta un gráfico elaborado por *Greenpeace* [4] que enumera los peores en España durante el último decenio.

Figura 1.5. Ranking de los peores incendios en España en el último decenio. Greenpeace.

De todos los nombrados, hay dos en la lista que se han producido recientemente, en 2017, por lo que se ofrece al lector un balance resumido de ambos a continuación con datos publicados por el periódico *El País* [5]:

- Nº14. Moguer (Huelva):

  El fuego duró 3 días, arrasó más de 8000 hectáreas de matorral y arbolado debido a la fuerza del viento y al calor. Obligó a desalojar a más de 2000 personas, sin víctimas mortales ni heridos. Fue sofocado a escasos kilómetros del Parque Nacional de Doñana. La factura económica del fuego asciende a unos 2 millones de euros, sólo en labores de extinción. Se desconocen las causas.

- Nº17. Encinedo (León):

  Provocado en la comarca leonesa de La Cabrera, arrasó más de 8000 hectáreas, pero no dejó ningún herido. Las laborales de extinción aéreas fueron primordiales para su reducción debido al difícil acceso al núcleo del incendio por tierra.

- Oleada incendios en Galicia:

Pese a no aparecer en el ranking, más de una treintena de incendios colapsaron Galicia a finales de octubre del 2017. Pontevedra, Ourense o Vigo, fueron algunos de los focos. Se cerraron colegios y se produjeron desalojos masivos. El balance fue más de 4000 hectáreas quemadas y 4 personas fallecidas por atrapamiento y en labores de extinción. Sin tener una respuesta definitiva todo apunta a pirómanos como causantes de éstos.

En la figura 1.6. se indica visualmente la distribución de los incendios alrededor de las ciudades más pobladas de Galicia.

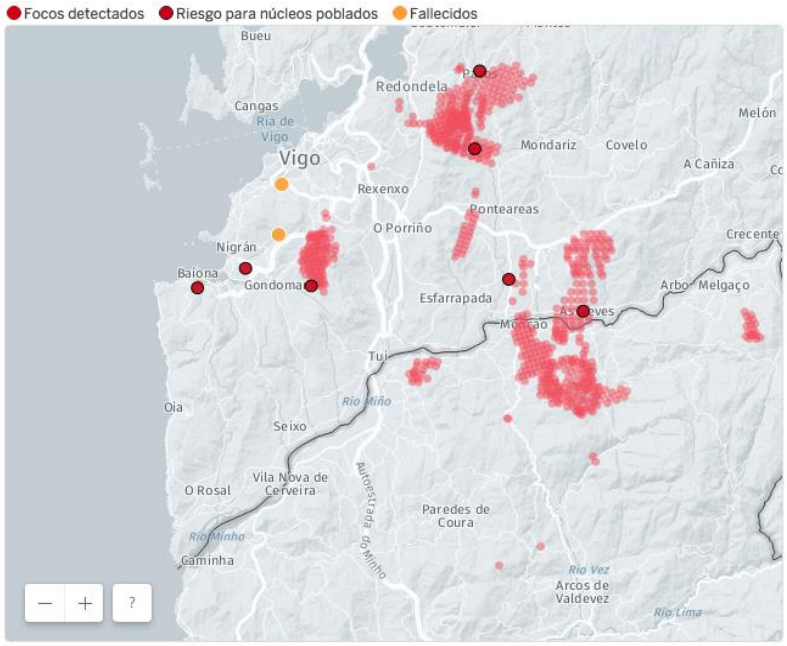


Figura 1.6. Principales focos en la oleada de incendios de Galicia en 2017

- Incendio de Pedrógão Grande (Portugal):

Por extensión tampoco se incluye en el ranking de la figura 1.5. pero debe quedar constancia del incendio en el que más pérdidas humanas se produjeron en un incendio forestal en la historia de Portugal. 65 víctimas mortales y 135 heridos. La mayoría de las víctimas fallecieron en una carretera dentro de sus coches ya que las llamas los acorralaron. Esto ocurrió en junio del 2017 cuando se llegaron a contar hasta 156 fuegos extendidos por todo el país en el momento más crítico.

1.1.5. El software de previsión de incendios forestales

Los periodos de análisis tras haber sufrido un incendio de gran magnitud son vitales para la próxima vez que se produzca uno, de tal manera que se puedan evitar errores o negligencias cometidas. Por otro lado, las acciones que han sido exitosas también se someten a juicio para poder ser aún más eficaces la siguiente vez. Estas mejoras en la actuación por pequeñas que sean, en todos los casos explicados en el apartado anterior pueden suponer un menor número de víctimas y daños materiales.

Durante esos momentos de reflexión, se estudian en primer lugar, las acciones que pueden evitar el incendio. El primer objetivo es implementar una cadena de acciones y procedimientos enfocados a la herramienta más eficaz, la prevención. Esta metodología engloba distintas opciones de actuación:

- Sensibilización de la ciudadanía: A través de campañas publicitarias en los medios de comunicación y principales canales públicos se muestran los elementos más impactantes de todo el proceso de extinción de un fuego para fijar lo mejor posible el mensaje en la mente del ciudadano.

- Exigentes castigos a los infractores: El límite de tolerancia se reduce al máximo donde en una escala de penalidad se dictaminan multas y penas imponentes para frenar a los delincuentes a atentar contra la naturaleza.

- Medidas indirectas: Tras prohibir por ley acciones que implican un riesgo directo de incendio, hay casos en los que se necesita aumentar un escalón más la precaución. Entonces se restringen comportamientos y escenarios que favorezcan la aparición de acciones que directamente produzcan incendios. Como casos ejemplificados se comenta lo siguiente. Tras haberse prohibido lanzar colillas de cigarros al campo (acción que provoca directamente incendios), se prohíbe fumar en el coche durante el periodo de conducción. Ya sea en zonas rurales o llevándolo a un nivel superior de exigencia, prohibirlo en cualquier carretera y lugar (acción que indirectamente puede producir incendios).

Sin embargo, cuando ya se ha provocado el incendio, la prevención ya no tiene efecto. Entonces, ¿qué soluciones tienen los brigadistas y todos aquellos profesionales encargados de reducir el fuego? ¿Los jefes deciden mirando un mapa cuál es el camino crítico y en qué zonas hay que situar más efectivos? Realmente, se puede intuir que hay margen de mejora en este punto.

Una herramienta de predicción que, tras establecer una serie de parámetros como *input* pueda calcular de manera fiable hacía que donde avanzará el fuego, reduciría el impacto negativo que provoca su incertidumbre de avance y dotaría a los servicios de protección de mayor poder e información para gestionar de una manera más eficaz sus recursos con el objetivo de eliminar por completo el número de fallecidos en estos desastres.

Cualquier modelo que trate de simular el avance del incendio debe cumplir con la máxima de calidad operativa y economía computacional. Es decir, menor complejidad técnica del sistema supone mayor velocidad de respuesta y menor coste. Lo que implica decisiones y ejecuciones físicas más rápido y la posibilidad de disponer de dicho programa a cualquier organismo de protección por muy básico que sea respectivamente.

1.2. Objetivo

Este Trabajo de Fin de Grado surge a partir de un programa que simula la propagación de incendios forestales escrito en lenguaje MatLab donde se aplica un algoritmo extraído del sector de la robótica, Fast Marching, utilizado para calcular trayectorias.

Sin embargo, este TFG no documenta la creación de dicho programa (simulador de incendios forestales) pues fue ya realizado por dos estudiantes previamente.

Adrián Bargueño, fue el primero en realizar un acercamiento del algoritmo al campo de la predicción de fuegos por medio de un programa en MatLab en su TFG [6]. Crea una interfaz de usuario de MatLab (GUI) en la que se lleva a cabo una simulación de la expansión del incendio en 2D, se puede elegir el origen y el final del fuego. Para su visualización en 3D, Adrián rellena un script a partir del código de la simulación en 2D.

Carolina Nicolás Martín mejora el programa de Adrián aportando un interfaz de usuario tridimensional, introduciendo el viento y la pendiente del suelo como parámetros de entrada por medio de matrices vectoriales, permitiendo al usuario elegir entre dos mapas con elevaciones reales, visualizando mapas de combustible y la propia simulación sobre el área de terreno seleccionado. Tratando de formar un interfaz tan sencillo que pueda utilizarlo cualquier ciudadano. Su excelente trabajo queda documentado en su memoria del TFG [7].

Éste es el punto de partida de este proyecto de investigación que, en la misma línea de desarrollo de ambos estudiantes, y teniendo como premisa la simplicidad de la ejecución del algoritmo Fast Marching, intenta mejorar la calidad del resultado del simulador.

Por tanto, este trabajo se basa en una mejora de este programa. La mejora es introducir la influencia de la orografía como elemento modificador sobre viento del incendio. En concreto, su dirección y velocidad.

Antes de este proyecto de mejora, pese a que se introduce un valor de viento como *input* del algoritmo, tal valor no tiene en cuenta la forma del terreno sobre el que se desarrolla el incendio. Cualquier posición del viento dentro del área seleccionada tiene la misma dirección y fuerza. Independientemente de si se encuentra en una llanura o justo al lado de una montaña. A grandes rasgos, podría considerarse correcta esta suposición. Sin embargo, es poco real porque en zonas de montaña el viento más cercano a la superficie se encuentra obstáculos terrestres que le obliga a modificar su avance y su velocidad se reduce. Pero en zonas de llanura, sin obstáculos en su avance, eso no sucede y su velocidad se mantiene constante y sin cambios abruptos de dirección.

Esta es la razón por la que esta mejora debe ser ejecutada. Aunque es específica, aporta veracidad e información crucial y acerca el simulador al objetivo de la línea global que siguen los Trabajos de Fin de Grado enfocados en este simulador: Crear un sistema real de predicción de incendios que llegue a los profesionales que combaten físicamente los fuegos y puedan extraer valor de la simulación mejorando sus actuaciones.

Debe ser mencionado que el hecho de poder entenderse intuitivamente la mejora y su importancia no implica que la implementación sea clara y sencilla.

1.3. Estructura

A continuación, se expone al lector el orden de los capítulos de cara a facilitar el entendimiento secuencial del proceso explicativo de este proyecto de mejora, a posicionar cada capítulo dentro de la estructura global y a destacar los más relevantes en la consecución de la mejora.

En el Capítulo 2 se estudian los modelos existentes de simulación de incendios junto a las herramientas antecedentes en dicha materia que dominan el mercado pese a que este sea reducido, sobre todo en España. El fin es conocer el estado actual de otras herramientas y los modelos para entender la importancia de llevar a cabo dicha mejora en el simulador con vistas a ser lo más competitiva posible.

En el Capítulo 3 se analizan los conceptos teóricos que envuelven el algoritmo de resolución empleado en la mejora y elemento resolutivo del simulador, Fast Marching.

El Capítulo 4 es el capítulo más importante porque se desarrolla la explicación pertinente de la implementación de la mejora en el simulador, detallando el plan resolutivo e indagando en el código de cada función utilizada para resolver la problemática, también descrita en este capítulo.

En el capítulo 5 se demuestra mediante representaciones gráficas de simulaciones realizadas el impacto de la mejora en el simulador como resultado final además de las conclusiones que genera éste.

Para el último capítulo, se establece una serie de líneas de mejora del simulador en las que realizar próximos trabajos para situar el programa entre los más destacados.

En el apéndice A se describe la situación del proyecto en relación a un marco regulador.

En el apéndice B se describe el entorno socio-económico de este proyecto junto a su presupuesto.

El apéndice C contiene el código íntegro de las funciones más importantes.

CAPÍTULO 2

ESTADO DEL ARTE

Han quedado demostrados la necesidad y el beneficio de ofrecer a los servicios de protección herramientas que simulen el avance del fuego para adelantarse a éste y reducirlo más rápido. Esto no es nuevo. El desarrollo de este tipo de recursos lleva ya años sucediendo, y en países como Estados Unidos, existen elementos reales de predicción que utilizan para sofocar los grandes incendios que sufren dentro de sus fronteras cada año.

Los pilares sobre los que se cementan son la sencillez computacional, la minimización de su coste de aplicación y la efectividad del algoritmo en base al menor número de datos analizados posible para favorecer una mayor velocidad de cálculo. Este algoritmo tiene el único mandato de obtener de la manera que sea la trayectoria de avance del incendio. Debe gestionarse adecuadamente la combinación de parámetros de entrada, pues el progreso de un incendio es dependiente de un alto número de parámetros. Viento, terreno, clima. Es por tanto fácilmente entendible la cantidad de distintas vertientes que se pueden implementar para obtener un algoritmo de predicción, en este caso de incendios forestales.

Para los próximos apartados, el lector recibe una explicación y estado actual de las más potentes herramientas existentes en el mercado con aplicación real, valorando sus puntos diferenciadores en relación a los demás.

2.1. Modelos físicos

Su base es la utilización de las leyes físicas como métodos de cálculo. En este caso, aquellas que aplican en lo referente a expansión de incendios. Las leyes de transporte molecular, como la Ley de Newton de la viscosidad, la ley de Fourier o la primera ley de Fick, que son las que explican la transferencia de cantidad de movimiento, energía y materia a nivel molecular, entrarían dentro de las Leyes Físicas aplicadas en los métodos físicos. También abarca modelos computacionales de dinámica de fluidos.

Los parámetros de entrada físicos se clasifican en 3 grupos mayoritarios:

- Orografía: Se incluyen la inclinación del terreno, la forma y el relieve de éste. La inclinación afecta porque de acuerdo con las leyes de convección del calor, el fuego tiende a avanzar hacia arriba más rápido (el aire caliente es menos denso que el aire frío, lo que produce corrientes de convección ascendentes), es por esto por lo que en una cuesta un incendio avanzará hacia arriba más que hacia abajo. La forma del relieve y su textura provocarán cambios en los vientos más próximos, que afectarán al movimiento y avance de las llamas.

- Combustibilidad del entorno: Aquellas características que provocan que los materiales ardan más fácilmente o menos. Pertenecen a este grupo la humedad y la tipología de la flora, es decir, si hay arboleda u arbustos que arden fácilmente. En

general, qué tanto por ciento de material potencialmente puede convertirse en combustible y medio propagador del fuego.

- Climatología: Es variable climatológica la temperatura debido a su influencia en la transmisión de calor y su consecuente implicación en las corrientes del aire. También lo es el viento, elementos influyentes en la dirección de avance del incendio y la sequedad del ambiente donde en función de la concentración de moléculas de agua se producirá una propagación más rápida o lenta.

Cada parámetro tiene su propia influencia y es individualmente analizada. Un ejemplo es el artículo de investigación [8], en el que se estudia cómo cambia la expansión del fuego con pequeñas variaciones del flujo del aire del lugar. Cambios que son generados por el movimiento del aire como consecuencia de la propia llama y por los vientos meteorológicos propios de la zona.

Sin embargo, no interesan modelos que analicen individualmente el efecto de una variable en la propagación. Sino introducir varias a la vez en el algoritmo, cuantas más mejor pues se acercaría más a la realidad, y obtener resultados a medida que cuenten con la influencia de todos esos parámetros. En contraposición, se incurrirá en mayor coste y tiempo de ejecución. Ser capaz de alcanzar el grado de precisión y complejidad o coste deseado marcará el éxito del programa resultante.

2.2. Modelos experimentales

Un modelo experimental es un modelo que se fundamenta en la lógica empírica, observación de fenómenos y análisis estadístico de los datos almacenados. En el caso de la propagación de un incendio se sacan conclusiones a partir incendios que han existido previamente o se ensaya con incendios provocados para tal fin que están completamente controlados y donde su alcance es previamente decidido.

En este tipo de modelos la influencia de una determinada variable en el resultado final no queda reflejada en ningún análisis, hecho que si ocurre en los modelos físicos. Sin embargo, mediante los experimentos puede analizarse cómo afectan en la propagación algunos parámetros como viento o temperatura.

Cuando los modelos físicos no consiguen establecer relación entre una determinada variable y el avance del fuego por motivos de complejidad o de capacidad computacional, se estudia si mediante un experimento podría quedar cubierto dicho análisis.

Principalmente existen dos tipos de experimentos:

- En terreno real: Al aire libre se realizan pequeñas fogatas y se apunta como se propaga. Por obligación, algunas variables son fijas, principalmente la temperatura. Otras sí varían y se puede reportar su influencia, por ejemplo, la dirección del viento, la cantidad de hogueras iniciales o la variación de origen del incendio. A pesar de que los resultados estadísticos no son los mejores en términos de calidad, se pueden realizar estudios con importante aportación científica como es el caso de [9].

- En laboratorio: Las aproximaciones son mucho más reales, ya que se pueden introducir más parámetros que en los incendios controlados experimentales en el campo. Sin embargo, su escala es mucho menor, esto provocada que todos los experimentos de laboratorio deban ser validados tras llevarse a cabo. Su fiabilidad es menor, pero su coste también, así como su ejecución es más fácil y segura. Esto lo demuestra el estudio [10].

En una comparación entre los dos primeros tipos de modelos, los modelos físicos son más costosos que los experimentales. Además, el acceso al dato en los experimentos es inmediato pues está almacenado. En términos económicos, los experimentos exigen más inversión que los físicos.

2.3. Modelos computacionales

Basados en la utilización de ecuaciones discretizadas. Para este trabajo son ecuaciones de expansión y se solventan con algoritmos aplicados a mallas discretas. El [11] es uno de los ejemplos de modelo más representativos y actuales. Cada celda de la malla tiene el mismo combustible, pero la distribución de los distintos tipos de combustible sobre el terreno se obtiene a partir de distribuciones probabilísticas y el método de muestreo de Monte Carlo. La silueta de las llamas y su avance están influidos por la capacidad de arder y por la sequedad que tenga cada posición del mallado. Usando un método dinámico de computación se puede calcular el mínimo tiempo posible en llegar a la próxima posición del mallado.

Además, se integra como parámetro el tiempo que tarda en apagarse el combustible desde el momento que el fuego ha sobrepasado esa posición del mallado, lo que se conoce como tiempo de resistencia. Está permitido también que se apague sólo, en el caso por ejemplo de que el combustible que está alrededor del frente del fuego esté mojado. El mayor inconveniente es que el viento no está presente en el algoritmo de resolución. Cuestión que lo aleja de la realidad.

2.3.1. Algoritmos Level Set y Fast Marching

Como resultado de los análisis realizados sobre la expansión de un incendio forestal, se puede constatar que la mínima distancia entre el punto inicial y final del

incendio no tiene por qué coincidir con el camino que sigue el incendio en su avance. Pero si se puede concluir que por ese camino es por donde menos tiempo tarde el incendio en llegar del punto inicial al punto final. Se propone calcular dicho camino por medio de Fast Marching, caso específico del modelo de conjunto de nivel. Dicho modelo es detalladamente explicado en el siguiente capítulo.

Se estudia la viabilidad de aplicar dicha metodología en la predicción de propagación de fuegos en [12], donde también se compara con otros casos parejos para determinar su éxito. En [13] se mejora el grado de cálculo con el empleo de métodos de nivel como el método Level Set (su influencia sobre el avance de la primera línea de fuego creando figuras como óvalos por medio de 3 elementos: cambios en el viento, velocidad y combustible).

Destaca el trabajo realizado por el profesorado y alumnado de la Universidad Carlos III de Madrid, en especial, el alumno Adrián Bargueño y la alumna Carolina Nicolás Martín después, por medio de sus respectivos Trabajos de Fin de Grado ([6] y [7]) junto con los profesores Santiago Garrido y Luis Moreno. Este proyecto de investigación es la continuación de sus proyectos del simulador con Fast Marching.

2.4. Modelos mixtos

Una vez más, pese a que cada uno de los modelos explicados pueden ser muy útiles, la combinación de éstos es aún mejor, ser capaz de ajustar restricciones de cada uno para extraer el beneficio de cada uno en un mismo algoritmo es potencialmente lo más cercano a la realidad. De esta manera, un modelo en el que se utilicen experimentos a la vez que físicos o de computación permitirá obtener conclusiones más precisas, completas y eficaces en la búsqueda de la simulación perfecta del movimiento del fuego. Otra característica diferenciadora de los modelos mixtos es su capacidad para evaluar la influencia de un parámetro de entrada desde los 3 modelos puros y decidir su validez en función de la semejanza que tengan los 3 resultados calculados. En caso de que sean semejantes, se podrá trabajar con datos con un alto nivel de fiabilidad.

Entre los modelos mixtos destaca [14] debido a su combinación de modelos matemáticos con empíricos en tubos de ventilación. Se incluye la influencia de las propiedades del material de combustión. Relación área/volumen de sus moléculas, extensión o niveles de porosidad del material. También las propiedades climatológicas y las de la topografía como la forma del suelo.

2.5. Herramientas de predicción destacadas

Tras evaluar los distintos procedimientos de modelaje de expansión de incendios forestales y aclarar que la elección de uno o varios varía con el número de variables que se quieran introducir en el algoritmo, es momento para reflexionar sobre otros aspectos igual de relevantes que el núcleo de la herramienta, los modelos y el algoritmo. En concreto, queda por definir cómo va a visualizar el usuario el programa cuando se disponga a utilizarlo (interfaz gráfica), cómo se van a obtener los datos procedentes de variables externas que son necesarios para el funcionamiento del sistema y el software que encaje en dichos requerimientos.

Estas tareas precisan esfuerzo colaborativo de profesionales de distintos sectores, donde se podría formar un equipo multidisciplinar que cuente con un experto en incendios forestales, un representante de los servicios de protección, programadores y matemáticos en aras de crear un escenario idóneo para lanzar la mejor versión posible del simulador.

Este proceso de construcción de simuladores ya ha sido realizado satisfactoriamente obteniendo los programas que se documentan a continuación, que, además, son los más relevantes en el mercado actual, por lo que representan perfectamente el camino a seguir de este Trabajo de Fin de Grado para alcanzar el nivel de competencia necesario.

2.5.1. FireRS

Se trata de un proyecto (Wildfire Remote Sensings) en proceso de mejora que debido a sus pilares esenciales es potencialmente una herramienta de gran uso en el futuro [15]. Propone dotar a toda organización de protección y centros de coordinación con una serie de datos a nivel instantáneo como contorno del fuego, visión infrarroja, coordenadas vía GPS, metodología de acción y previsión del avance del fuego por medio de un completo y novedoso instrumento electrónico.

La estructura técnica cuenta con un microsatélite de transmisión, drones, sensores de luz infrarroja y un sitio de coordinación para gestionar la integración y funcionamiento de cada elemento. Con una duración de 3 años (con fecha de corte julio del 2019) y un presupuesto de 2 millones de euros, ha atraído a potentes inversores como el Fondo Europea de Desarrollo Regular o el Centre National de la Recherche Scientifique.

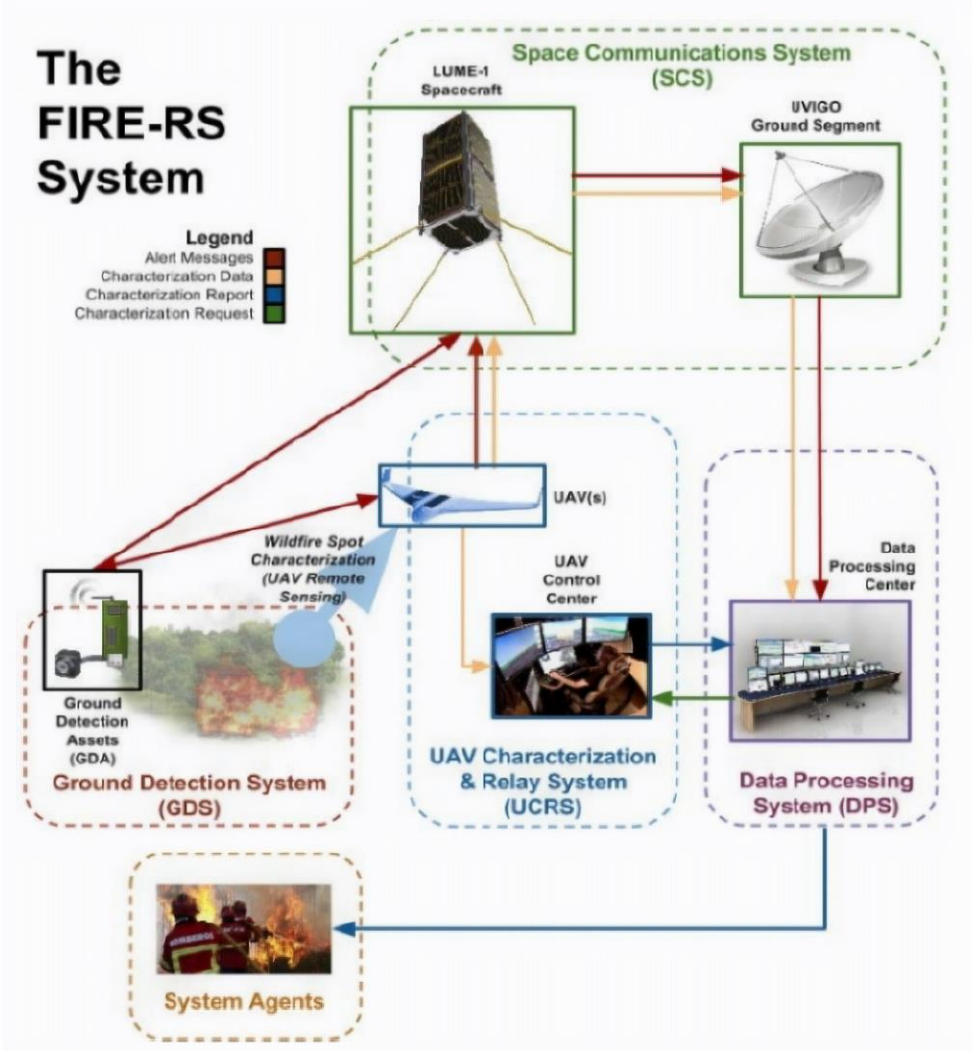


Figura 2.1. Esquema de FireRS

2.5.2. Prometheus

Sistema de predicción de incendios forestales más empleado en Canadá [16]. Parámetros de orografía como la altura o inclinación del terreno, parámetros de combustibilidad y climatológicos conforman el *input* del modelo computacional de Prometheus basado en el principio de Huygens.

Tal y como explica *VARINIA.ES* en su blog [17], el principio de Huygens "permite predecir la posición futura de un frente de onda cuando se conoce su posición anterior". En la misma línea de argumentación, afirma que este principio "establece que los frentes de onda están formados por frentes de onda más pequeños, es decir, que cada punto de un frente de onda primario se comporta como un emisor de ondas secundarias esféricas, con igual frecuencia y que se propagan en todas las direcciones con la misma velocidad que la primaria en cada punto". Por último, concluye que "La envolvente de todas esas ondas secundarias en el nuevo frente de onda formado".

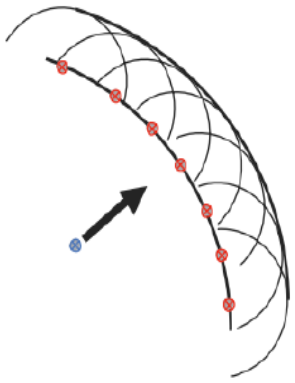


Figura 2.2. Nuevo frente de onda según el principio de Huygens

Se comienza por asociar a la línea de avance del incendio una figura geométrica cuyas esquinas lanzan una expansión de onda distinta, acorde con las propiedades de esa esquina (combustibilidad, orografía o climatología).

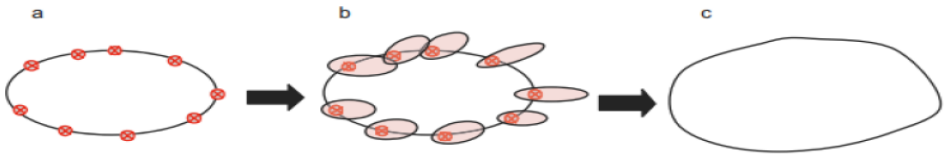


Figura 2.3. Representación gráfica de formación del nuevo frente de onda en base al principio de Huygens

En lo referente a software, Prometheus emplea una aplicación de programación de Microsoft (COM – *Common Object Model*) cuya principal funcionalidad es la migración de componentes orientados a objetos a otros entornos de programación sin necesidad de compartir el código original.

Prometheus es utilizado como código principal por algunos softwares con generación de interfaz gráfica de incendios forestales como es el ejemplo de Burn-P3 [18], donde los parámetros de entrada (viento, área montañosa seleccionada, climatología, mapas orográficos de distintos juegos de colores o información del combustible como parámetro obligatorio) entran con lenguaje ASCII a Burn-P3 teniendo como resultado la siguiente figura:

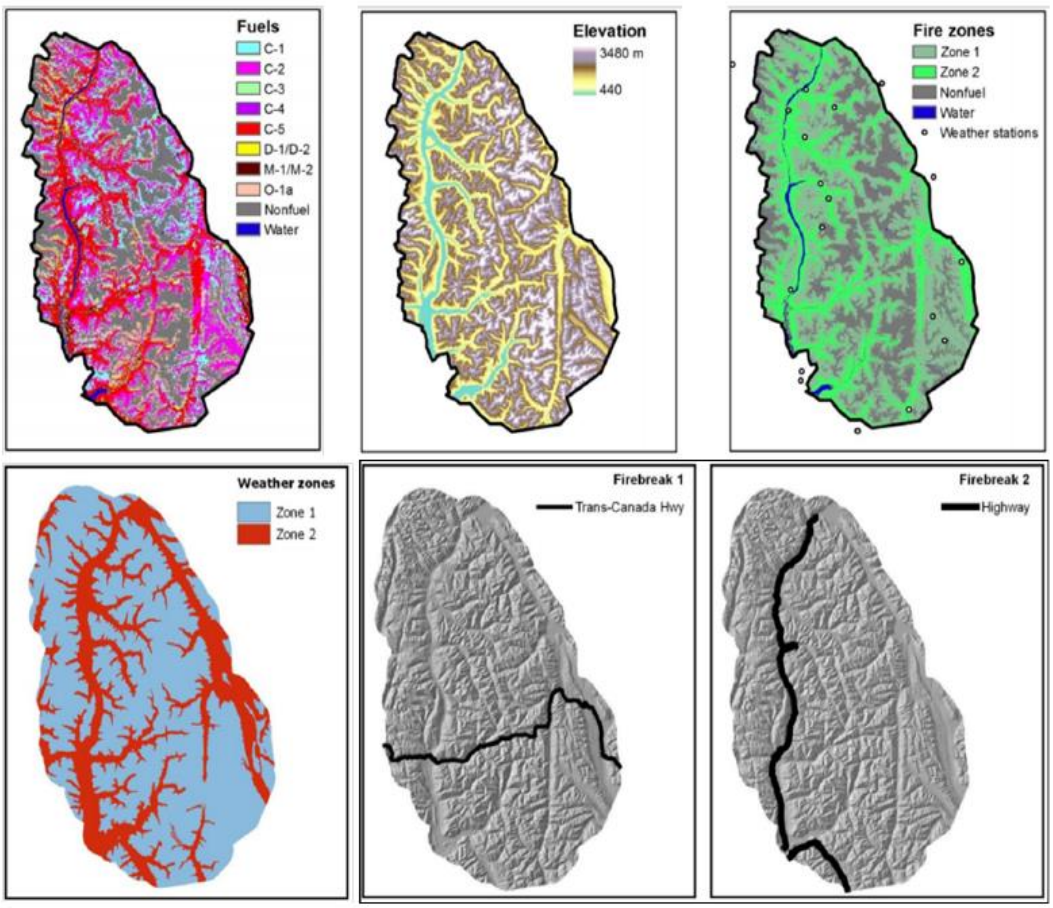


Figura 2.4. Distintos casos de *inputs* en Burn-P3

El resultado producido por Burn-P3 está compuesto por 3 documentos:

- Mapa que indica la probabilidad de que se queme un área seleccionada por medio de colores, como se observa en la figura 2.5.
- Análisis estadístico del mapa
- Archivo tipo log con el resumen de la acción computacional

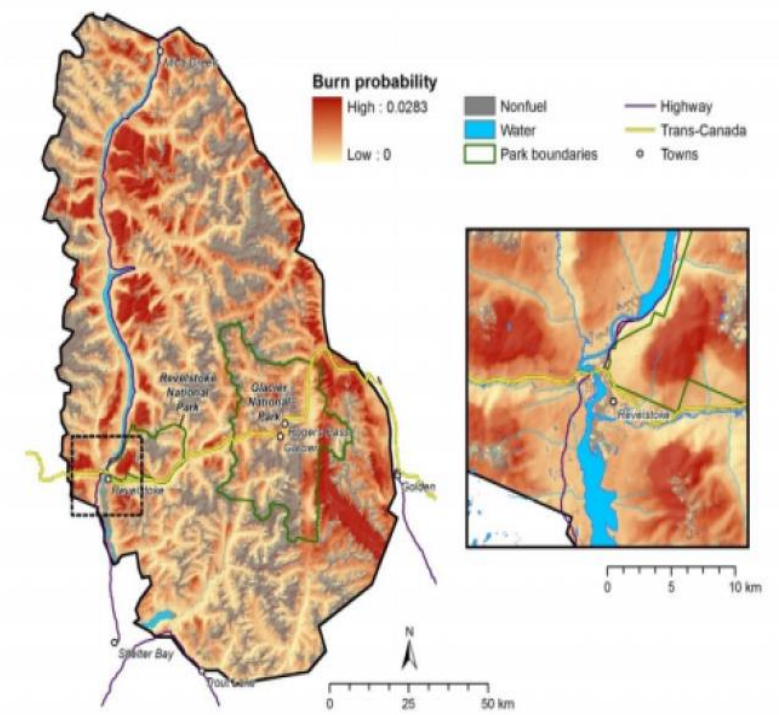


Figura 2.5. Representación de la probabilidad de que se queme el terreno

2.5.3. FIRESTAR

Se trata de un sistema de soporte a decisiones sobre la gestión de combustible y reducción del peligro de los incendios en lugares rurales y urbanos de área mediterránea. Entre sus objetivos, destaca el perfeccionamiento de modelaje del combustible floral mediterráneo y de la precisión predictiva de propagación del incendio.

Su sistema predictivo se basa en las leyes físicas [19] en el que se solventa un grupo de ecuaciones diferenciales parciales de la conservación de momento, masa y energía. FIRESTAR es capaz de analizar la correlación entre dos parámetros iniciales y además con el avance del fuego. Se pueden observar en la figura 2.6. estas relaciones. Por ejemplo, las ráfagas de aire (U_h) en relación con la temperatura y la velocidad (Visualizado en rango de colores y mapa de vectores, respectivamente).

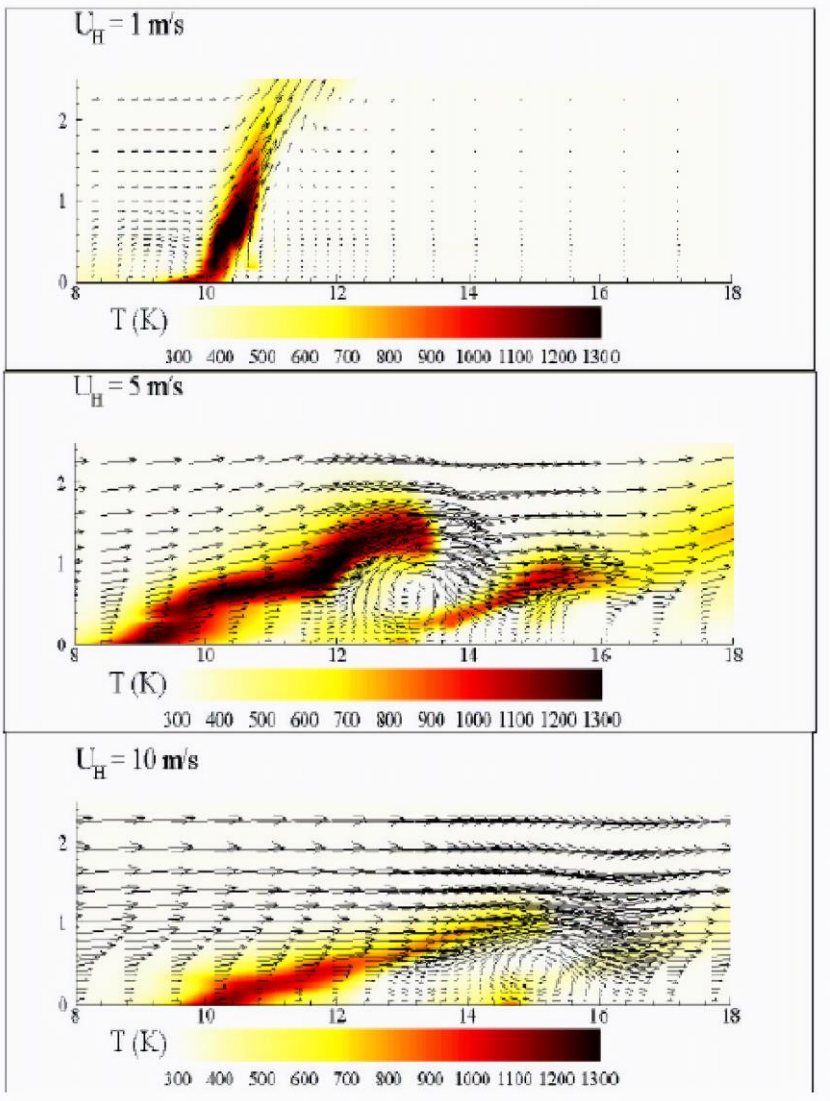


Figura 2.6. Variación de la temperatura y velocidad del gas a distintas velocidades de viento

El contratiempo de FIRESTAR es su incapacidad para representar sus resultados en 3D ya que el hecho de trabajar exclusivamente con un modelo físico puro obliga a tener un coste computacional elevado que permite como máximo su visualización bidimensional.

2.5.4. FARSITE

Es el programa que mejor predice del mercado la propagación de un fuego [20]. Se sostiene sobre el principio de Huygens, al igual que el simulador canadiense

Prometheus (apartado 2.5.2.), pero además se apoya también sobre el modelo de Rothermel y el inicio de fuego de copas de Van Wagner.

El esquema de la figura 2.7. representa el sistema de acción donde los cuadros grises marcan los pasos que debe seguir el usuario, las carpetas amarillas señalan los formatos de documento que puede expulsar el programa en cada paso. Los parámetros de entrada también aparecen indicados tanto si son obligatorios o no (climatología, combustible o elevaciones del terreno en forma de mapa).

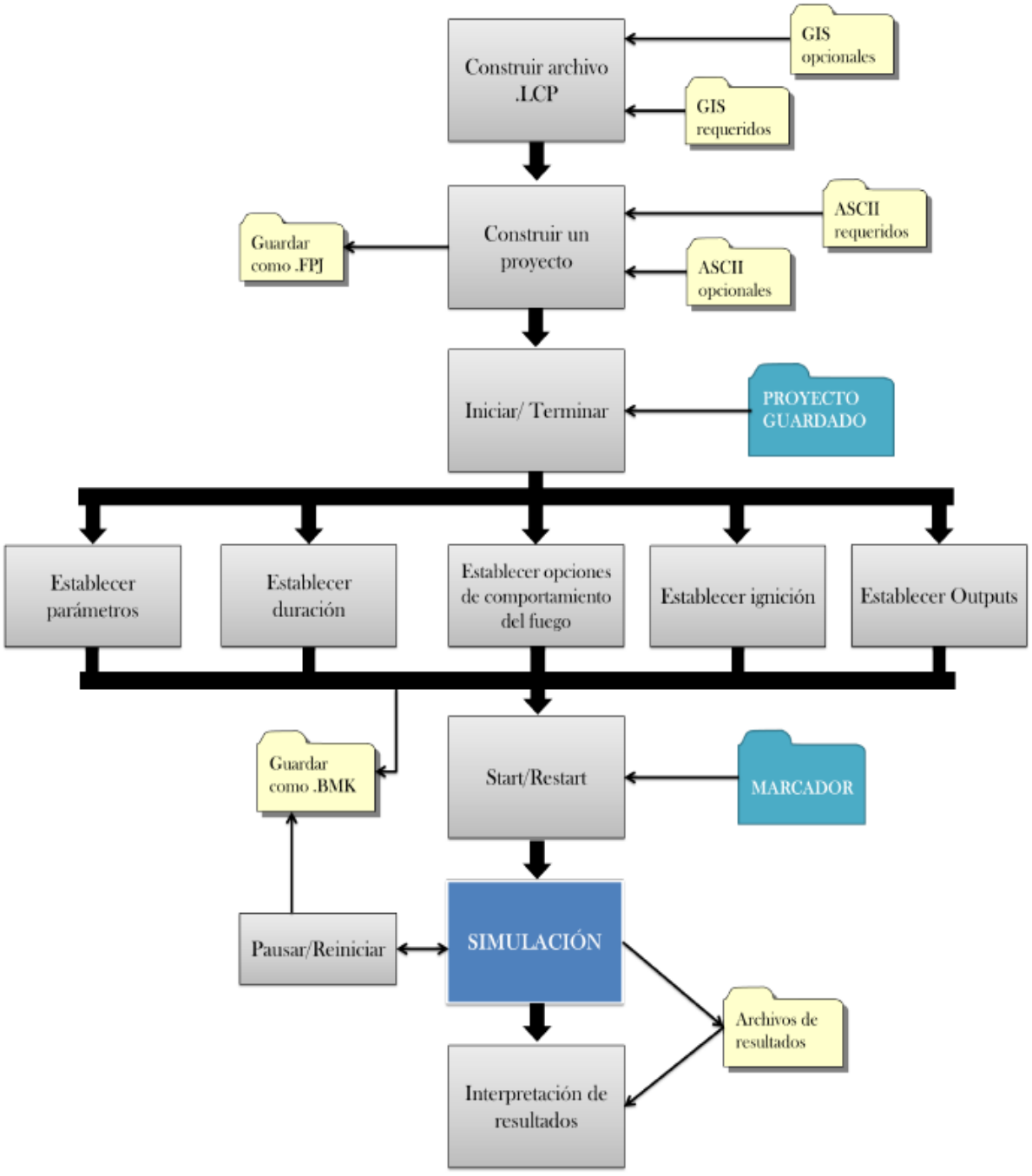


Figura 2.7. Diagrama del procedimiento de actuación de FARSITE

Para la comprensión del lector, en la figura 2.7. está representado un ciclo de simulación del programa FARSITE [21]. Su éxito es debido a la cantidad de parámetros que absorbe el programa, así como a la calidad y óptica global de su *output* ya que se generan multitud de referencias y relaciones entre variables y propagación. La parte menos competitiva es la visualización del avance del incendio. Las líneas que lo representan no son claramente diferenciadas del terreno por tanto la información no se transmite adecuadamente. Por otro lado, el acceso al cambio de tono de los distintos parámetros lejos de ser intuitivo se complica demasiado, factor que en este proyecto de investigación toma gran importancia, se considera que la sencillez de la pantalla del usuario debe ser lo más universal y clara posible.

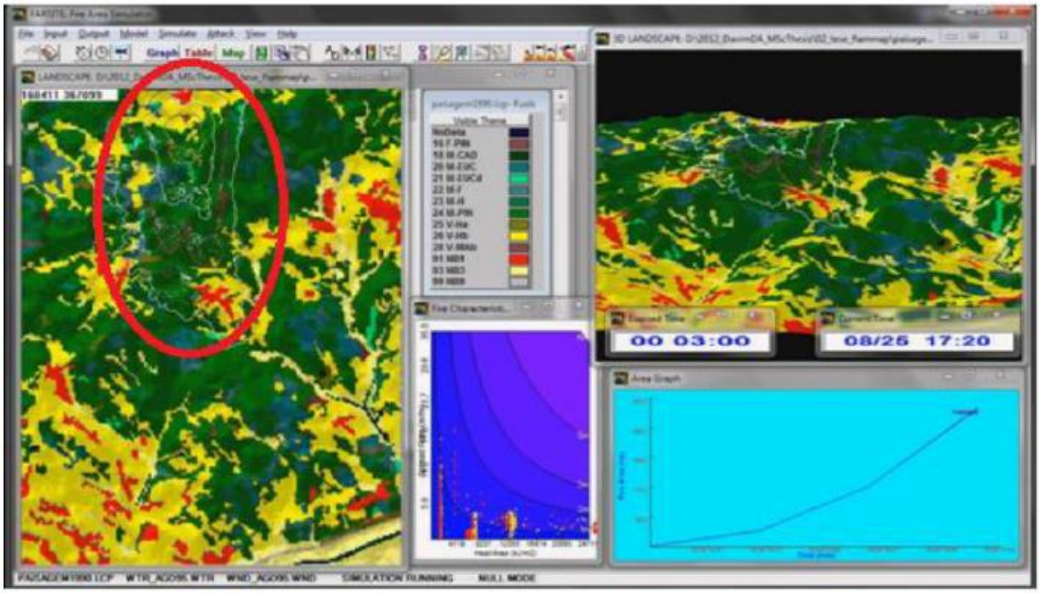


Figura 2.8. Pantalla de FARSITE tras ejecutar un ciclo de simulación

CAPÍTULO 3

METODOLOGÍA

En este tercer capítulo se desarrolla una línea de argumentación matemática y científica sobre la inclusión de Fast Marching, algoritmo creado específicamente para robótica, en la simulación de incendios forestales.

Fast Marching (FM) nace del método *Level Set* y su finalidad inicial es recrear la expansión de una onda es un volumen discreto. Su potencial radica en la capacidad para solventar la ecuación Eikonal en la expansión de ondas.

Los apartados del capítulo se ordenan de la siguiente manera:

- 3.1. Conocimiento general del algoritmo FM
- 3.2. Resolución de la ecuación Eikonal según [22]
- 3.3. y 3.4. Casos de aplicación del algoritmo de Dijkstra a la resolución de FM
- 3.5. Efecto de una matriz vectorial ajena sobre FM

3.1. Conocimiento general de Fast Marching

Para comprender el funcionamiento de este algoritmo se propone al lector el ejercicio de visualizar mentalmente una bola de granizo impactando en una piscina y las respectivas ondas que son producidas como resultado. Es importante que dicha situación esté presente durante todo el capítulo ya que ayudará a comprender mejor que la forma de las ondas en el agua, el tiempo que se pueden observar y sus velocidades dependen (contando que variables externas al caso como la velocidad de caída del granizo es siempre el mismo) del fluido que contenga la piscina.

Si la piscina contiene únicamente agua el contorno de las ondas será circular con un tiempo de observación y velocidad determinadas. Si se añaden otros líquidos a la piscina como aceite en una proporción del 50%, el frente de expansión de las ondas dejará de ser una circunferencia para convertirse en una forma irregular. Además, cambiará la velocidad y el tiempo que se puedan observar las ondas hasta llegar a otro punto de impacto.

Se puede concluir que su propagación depende de la viscosidad del fluido, lo que en un nivel mayor de profundidad, significa que depende del área de velocidades en el que se expande. La trayectoria se obtendrá aplicando un descenso del gradiente en el que se tomará su variación mayor.

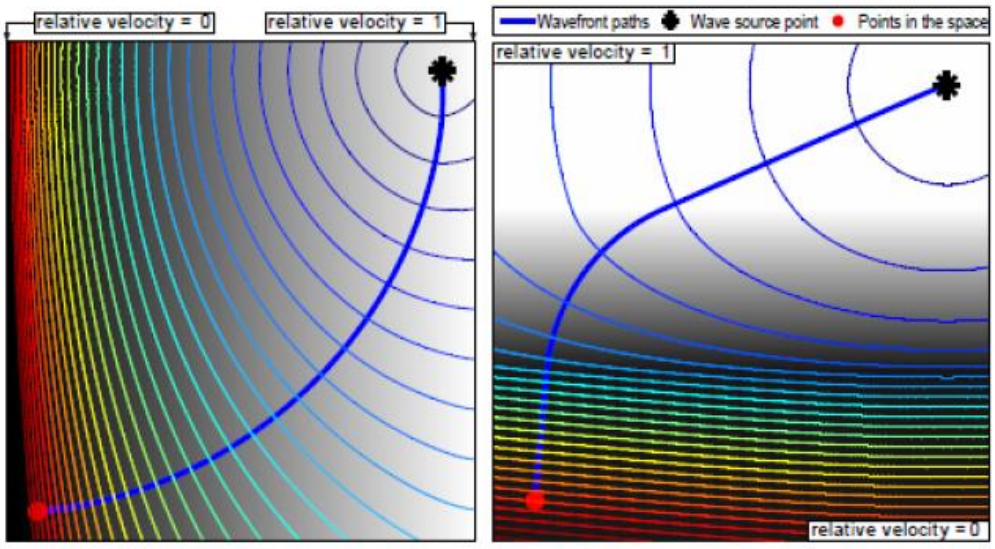


Figura 3.1. Expansiones de onda en distintos campos de velocidades

Se procede a enumerar algunas puntualizaciones importantes previa explicación matemática de la ecuación Eikonal:

- El comportamiento de un frente de onda es el caso extrapolado del avance de la primera línea de fuego de cualquier incendio en un entorno científico y matemático, de ahí la posibilidad de trasladar los resultados obtenidos en la resolución de ecuaciones de ondas a la posible manera de propagarse el fuego. En otras palabras, se pacta que una onda actúa igual que un incendio en su expansión dentro de las hipótesis establecidas durante este estudio técnico.

- Al utilizar el algoritmo FM para simular incendios obligatoriamente se descarta la posibilidad de éstos de retroceder. Esto es causa de considerar un mapa de velocidades siempre positivo. Esta asunción no supone graves inconvenientes pues la mayoría de los eventos físicos de esta índole ocurre esto. Igualmente, no hay incendios que retrocedan ya que físicamente es muy difícil que sobrevivan avanzando por terreno ya chamuscado y sin materia viva (suponiendo que la influencia del viento es nula). A pesar de estos argumentos, sí se producen retrocesos en algunos incendios. En estos casos se emplea el método Level Set en lugar de Fast Marching.

3.2. Resolución de la ecuación Eikonal

3.2.1. Descripción de la ecuación Eikonal

Seleccionamos un dominio $\chi \subset \mathbb{R}^N$ y una función velocidad en χ que agrupa las velocidades locales de la onda en cada instante, $F_{(x)}$. Para conseguir que un conjunto de valores iniciales $\chi_s \subset \chi$ lleguen a unos valores finales $\chi_{final} \subset \chi$ lo antes posible, la ecuación Eikonal establece el tiempo de llegada a los puntos del dominio $T_{(x)}$ bajo la siguiente restricción:

$$\left|\nabla T_{(x)} F_{(x)}\right| = 1 \tag{3.1}$$

Que si $\chi_s \subset \chi$:

$$T_{(x)} = 0 \tag{3.2}$$

En busca de mayor comprensión piense el lector en la forma unidimensional donde $\left|\nabla T_{(x)}\right|$ es un gradiente:

$$\left|\nabla T_{(x)}\right| = \frac{dT}{d\theta} \tag{3.3}$$

Con θ como valor de distancia. Así se puede expresar la ecuación de la velocidad de una manera clara:

$$\theta = FT \tag{3.4}$$

Derivando la ecuación (3.4) con respecto a θ y empleando (3.3) se consigue:

$$1 = F|\nabla T| \tag{3.5}$$

3.2.2. Discretización de la ecuación Eikonal

Un mallado cartesiano con las funciones discretizadas F y T representa el dominio, que es un cubo de N dimensiones. Sin embargo, para la simplificación de la demostración se procede a establecer N=2. Ahora, F y T son discretizadas en un mapa de posiciones. $T_{i,j}$ y $F_{i,j}$. Mapa de la función tiempo y velocidad respectivamente.

Se explica la discretización de orden 1 más común de la ecuación Eikonal trabajada por Sethian J.A y Osher S. [23].

$$\begin{Bmatrix} \max\left({D_{ij}}^{-x} \cdot T, 0\right)^2 + min\left({D_{ij}}^{+x} \cdot T, 0\right)^2 \\ \max\left({D_{ij}}^{-y} \cdot T, 0\right)^2 + min\left({D_{ij}}^{+y} \cdot T, 0\right)^2 \end{Bmatrix} = \frac{1}{{F_{i,j}}^2} \tag{3.6}$$

Se permite igualar las derivadas parciales de T según un *upwind scheme*, métodos de discretización numéricos que resuelven ecuaciones diferenciales parciales hiperbólicas. Por tanto:

$$\begin{aligned} \frac{\partial T}{\partial x} &\approx {D_{ij}}^{\pm x} \cdot T = \frac{T_{i+1,j} - T_{i,j}}{\pm \Delta x} \\ \frac{\partial T}{\partial y} &\approx {D_{ij}}^{\pm y} \cdot T = \frac{T_{i,j+1} - T_{i,j}}{\pm \Delta y} \end{aligned} \tag{3.7}$$

Donde ${D_{ij}}^{\pm x}$ es la derivada parcial numérica lateral en la dirección $\pm x$ y, Δx y Δy el espaciado de la malla en dirección x e y respectivamente.

Este procedimiento no es la única alternativa posible para escribir la ecuación Eikonal en 2D. La siguiente es igualmente válida, aunque menos eficiente:

$$\begin{Bmatrix} \max\left({D_{ij}}^{-x} \cdot T, -{D_{ij}}^{+x} \cdot T, 0\right)^2 \\ \max\left({D_{ij}}^{-y} \cdot T, -{D_{ij}}^{+y} \cdot T, 0\right)^2 \end{Bmatrix} = \frac{1}{{F_{i,j}}^2} \tag{3.8}$$

Y al sustituir (3.7) en (3.8) y sumiendo:

$$\begin{aligned} T &= T_{i,j} \\ T_x &= \min(T_{i+1,j}, T_{i-1,j}) \\ T_y &= \min(T_{i,j+1}, T_{i,j-1}) \end{aligned} \tag{3.9}$$

Se obtiene dicha ecuación:

$$\max\left(\frac{T - T_x}{\Delta x}, 0\right)^2 + \max\left(\frac{T - T_y}{\Delta y}, 0\right)^2 = \frac{1}{{F_{i,j}}^2} \tag{3.10}$$

Como previamente se ha especificado que los únicos valores posibles que pueden adquirir las velocidades de la línea de avance del incendio son aquellos que sean positivos ocurre que:

$$\begin{aligned} T &> T_x \\ T &> T_y \end{aligned} \tag{3.11}$$

Esto sucede para los casos en los que la onda no haya alcanzado la posición (i, j).

(3.11) permite simplificar de la ecuación Eikonal para 2D en (3.10):

$$\max\left(\frac{T-T_x}{\Delta x}\right)^2 + \max\left(\frac{T-T_y}{\Delta y}\right)^2 = \frac{1}{{F_{i,j}}^2} \tag{3.12}$$

3.2.3. Solución de la ecuación Eikonal

La ecuación (3.12) será representada como una ecuación polinómica de 2º grado:

$$\begin{aligned} a &= \Delta x^2 + \Delta y^2 \\ b &= -2\left(\Delta y^2 T_x + \Delta x^2 T_y\right) \\ c &= \Delta y^2 {T_x}^2 + \Delta x^2 {T_y}^2 - \frac{\Delta x^2 \Delta y^2}{{F_{i,j}}^2} \end{aligned} \tag{3.13}$$

Para la forma $aT^2 + bT + c = 0$.

Es posible escribir el desarrollo de 2D en su forma general, es decir, para N dimensiones.

Asignando el mismo valor a la separación de cada posición de los 3 ejes cartesianos:

$$\Delta x = \Delta y = \Delta z = \cdots = h \tag{3.14}$$

Junto con la nueva denotación de T_x y T_y como T_d debido a que se puede elegir cualquier dimensión *d*, se obtendrá la siguiente ecuación polinómica de 2º grado:

$$\begin{aligned} a &= N \\ b &= \sum_{d=1}^{N} T_d \\ c &= \left(\sum_{d=1}^{N} {T_d}^2\right) - \frac{h^2}{F^2} \end{aligned} \tag{3.15}$$

Si se desea sustituir N por 2, se obtiene:

$$\begin{aligned} a &= 2 \\ b &= -2\left(T_x + T_y\right) \\ c &= \left({T_x}^2 + {T_y}^2\right) - \frac{h^2}{F^2} \end{aligned} \tag{3.16}$$

Para la forma $aT^2 + bT + c = 0$. Y con solución general:

$$T = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{3.17}$$

En la que introduciendo los valores de (3.16) se tiene que:

$$T = \frac{T_x + T_y}{2} \pm \frac{\sqrt{\frac{2h^2}{F^2} - \left(T_x - T_y\right)^2}}{2} \tag{3.18}$$

Es importante mencionar que para alcanzar una posición del mallado alejado, debe ir pasando por posiciones intermedias o próximas. Poco a poco, hasta llegar a la deseada. Esto es explicado por el principio de causalidad. En caso de que dicha condición no fuera satisfecha, se incurriría en el error de respaldar que, para una determinada posición de la línea de avance, la velocidad sería negativa, situación que como previamente se ha advertido no se permite en la aplicación del algoritmo FM, pues todos los valores deben ser positivos.

Por tanto, (3.18) sólo será válido para FM en los casos en los que:

$$T \geq |T_x - T_y| \tag{3.19}$$

Dicha restricción, aplicada a (3.18) se traduce en la siguiente condición:

$$|T_x - T_y| \leq \frac{h}{F} \tag{3.20}$$

Para aquellos casos en los que la condición (3.20) no sea satisfecha, a la variable T se le adjudica un valor que no depende de T_x y T_y a la vez, sino de la que tiene el mínimo valor. Se conoce como *one sided update* y se aplica mediante la siguiente ecuación:

$$T = min\left(T_x, T_y\right) + \frac{h}{F} \tag{3.21}$$

3.3. Algoritmo de Dijkstra

3.3.1. Fundamento teórico

En este punto ya se ha presentado la base fundamental del algoritmo en torno al cual gira el proyecto, FM. Además, se ha descrito el proceso entero de resolución de la ecuación Eikonal, que es una pieza clave de FM.

Sin embargo, se puede mejorar el análisis general del algoritmo. En esta línea se procede a analizar el algoritmo de Dijkstra usando como referencia [24].

El objetivo de este fundamento teórico es el almacenamiento del conjunto de distancias que hay desde cualquier nodo de un mallado discretizado a los puntos de inicio.

Dicho mapa será llamado D. Por otro lado, para alcanzar cada nodo existe un coste de distancia que también debe ser almacenado en otra matriz llamada W. Con estas designaciones y el propósito de establecer una relación de la separación que existe entre una posición aleatoria j, se obtiene, teniendo como referencia la distancia de j con un nodo cercano k que:

$$D_j \leftarrow \min_{k \sim j} D_k + W_j \tag{3.22}$$

Se establece una restricción de clasificación de los nodos que permite llevar a cabo una correcta ordenación de los valores que se van almacenando en la matriz D. Cada nodo sólo puede pertenecer a uno de los 3 siguientes grupos:

- Desconocidos: Posiciones fuera de la expansión de la onda. Su valor actual es infinito a espera de ser alcanzado por la expansión.

- Banda estrecha: Aquellas posiciones que están a falta de 1 iteración para ser superadas por el frente de onda. Su valor métrico ya está computado, pero todavía no es fijo y en próximos ciclos de avance podría ser modificado.

- Congelados: Aquellas posiciones que han sido superadas por la expansión de la onda y por tanto su valor métrico ya está computado. Ese valor ya es invariable.

Desde que comienza el proceso de iteración todo nodo sufre el mismo trascurso de transformación:

Desconocido ⇢ Banda estrecha ⇢ Congelado

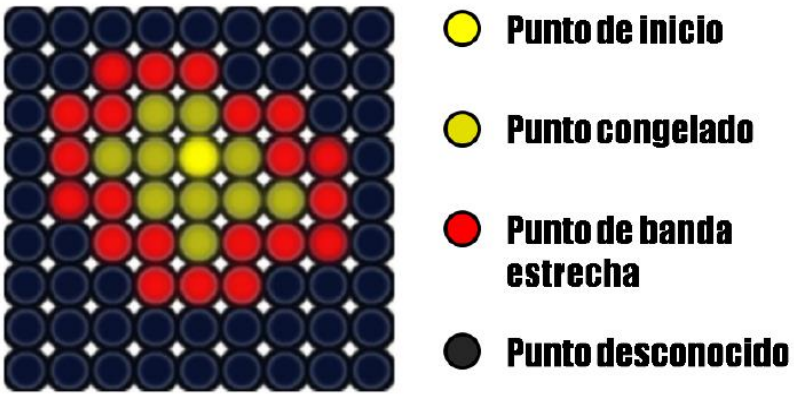


Figura 3.2. Visualización de los tipos de nodos según el algoritmo de Dijkstra

3.3.2. Caso práctico

Se procede a rellenar con valores ejemplificativos las matrices previamente descritas teniendo en cuenta que el objetivo es obtener el mapa de distancias D. En el apéndice C, se provee al lector con un código C.1. del algoritmo de Dijkstra de un caso simple.

- Matriz W: El mapa de costes métricos tiene un valor de 1 en todas sus posiciones para la simplificación de cálculo. Al ser un parámetro de entrada, W no varía durante el transcurso del proceso iterativo.

- Matriz S: Dentro de esta matriz que varía con cada repetición, los nodos congelados vienen representados por (-1), los de banda ancha por (1) y los que no han sido alcanzados por el frente de onda por (0).

- Matriz D: Todos los puntos tienen valor infinito en la iteración 1.

Se ofrece al lector una tabla de iteraciones en el proceso previamente detallado con matrices cuadradas de 3 filas y para el caso de 2D.

Número de iteración	0	1	2	3
Matriz S	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ -1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} -1 & 1 & 1 \\ 1 & 0 & 0 \\ -1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} -1 & 1 & 1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \end{pmatrix}$
Matriz D	$\begin{pmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ 0 & \infty & \infty \end{pmatrix}$	$\begin{pmatrix} 1 & \infty & \infty \\ 1 & \infty & \infty \\ 0 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 2 \\ 1 & \infty & \infty \\ 0 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 2 \\ 1 & 2 & 2 \\ 0 & 1 & 1 \end{pmatrix}$

Tabla 3.1. Conjunto de iteraciones en un caso sencillo de aplicación del algoritmo de Dijkstra

3.4. Algoritmo de Fast Marching

3.4.1. Fundamento teórico

Reagrupando los desarrollos anteriores, surge la comparación entre las distintas maneras presentadas para obtener distancias de nodos. Las dos opciones explicadas son Fast Marching o algoritmo de Dijkstra. Para el primero, se explica la resolución de la ecuación Eikonal discretizada (apartado 3.2.3.). Para el segundo, se observa su proceso de cálculo en el ejemplo de la tabla 3.1. La conclusión es que el procedimiento de Dijkstra es incompetente sin necesidad de compararlo con otro método cualquiera ya que el número de iteraciones necesarias para obtener resultados concluyentes es demasiado elevado.

Sin embargo, la solución final debe ser construida con ayuda del algoritmo de Dijkstra, por tanto, la siguiente tarea es implementar una copia de las matrices W y D procedentes de dicho algoritmo en el proceso resolutivo de la ecuación Eikonal. Se establece el siguiente paralelismo:

- La matriz D está asociada al parámetro tiempo, puesto que distancia y tiempo son directamente proporcionales.
- La matriz W se corresponde al quebrado de la distancia entre posiciones entre sus velocidades, es decir, $\frac{h}{F}$.

Matemáticamente se expresa de la siguiente manera:

$$D_k = \begin{cases} \dfrac{d_x + d_y + \sqrt{2{W_k}^2 - (d_x - d_y)^2}}{2} & si\ |d_x - d_y| \le W_k \\ \min(d_x, d_y) + W_k & para\ cualquier\ otro\ caso \end{cases} \qquad (3.23)$$

Teniendo que:

$$\begin{aligned} d_x &= \min(D_{k+1,l}, D_{k-1,l}) \\ d_y &= \min(D_{k,l+1}, D_{k,l-1}) \end{aligned} \qquad (3.24)$$

3.4.2. Caso práctico

Al igual que en el apartado 3.3.2. se propone un ejemplo sencillo como apoyo a la comprensión del algoritmo FM. En el apéndice C, se provee al lector con un código C.2. del algoritmo de Fast Marching de un caso simple.

En lo que respecta al procedimiento planteado en este apartado 3.4., el procedimiento es el mismo que en el 3.3. con la salvedad de emplear la ecuación (3.23)

en lugar de la ecuación (3.22) cuando se procede a calcular la próxima posición de la matriz D.

Se ofrece al lector una tabla de iteraciones en el proceso previamente detallado con matrices cuadradas de 3 filas.

Número de iteración	**0**	**1**	**2**	**3**
Matriz S	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ -1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} -1 & 1 & 1 \\ 1 & 0 & 0 \\ -1 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} -1 & 1 & 1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \end{pmatrix}$
Matriz D	$\begin{pmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ 0 & \infty & \infty \end{pmatrix}$	$\begin{pmatrix} 1 & \infty & \infty \\ 1 & \infty & \infty \\ 0 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1{,}7071 & 1{,}7071 \\ 1 & \infty & \infty \\ 0 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1{,}7071 & 1{,}7071 \\ 1 & 1{,}7071 & 1{,}7071 \\ 0 & 1 & 1 \end{pmatrix}$

Tabla 3.2. Conjunto de iteraciones en un caso sencillo de aplicación del algoritmo de Fast Marching

CAPÍTULO 4

DESARROLLO DEL TRABAJO

Todo programa nace de un mecanismo resolutivo computacional que resuelve un problema que beneficia de algún modo a la sociedad. Tiene sentido, por tanto, reconocer que es relevante la calidad del código, la funcionalidad y la sencillez de éste para que tenga éxito. Pero no es menos importante el plan elegido para mostrar ese valor generado al usuario o cliente. La pantalla y pasos que debe seguir el usuario para hacer funcionar la aplicación debe ser igual de competitivo y exigente que el algoritmo base.

La franja de estudio leída hasta aquí ha seguido la vertiente teórica del proyecto. Se ha tratado de situar y ordenar los distintos niveles teóricos de la aplicación para que se entiendan las relaciones y restricciones de cada componente teórico que hace posible el funcionamiento del simulador de cara a entender el escenario global en el que se va a implementar la mejora del programa.

Primeramente, se ha contado la importancia del porqué. La necesidad de crear un simulador de incendios. Los motivos físicos y sociales. La introducción. A continuación, el estado actual del sector en el que se va a crear el programa. La tipología de los sistemas existentes de predicción de fuegos. El estado del arte. Por último, la metodología seleccionada para llevar a cabo el simulador propio. La base matemática del algoritmo elegido para la validación del lector. Y su coherencia con la casuística que resolverá, llevar a cabo la mejora con la que incluir la influencia de la orografía en el viento durante un incendio. Fast Marching es el algoritmo seleccionado a pesar de ser normalmente utilizado en robótica. La justificación está desarrollada en el capítulo 3. Metodología.

En este cuarto capítulo, se procede a describir cada elemento de la interfaz de usuario en el entorno donde se construye el código del simulador (Matlab – [25]), lo estrictamente necesario para generar una ideal global al lector sobre el conjunto de elementos que debe conocer el usuario para utilizar el simulador. De la misma manera se cuenta cómo funciona el programa previo a la mejora. Ambos desarrollos se realizan a alto nivel ya que no forman parte directa de la materia de este trabajo, pero sí se consideran información clave para alcanzar un nivel de conocimiento del programa óptimo de cara a comprender la implementación de la mejora, que es el objetivo directo de este proyecto. La implementación de la mejora conforma la tercera y más extensa sección de este capítulo puesto que es la explicación del trabajo propio del alumno.

Pese a incurrir en una posible repetición, el alumno quiere dejar claro que este TFG no explica la creación del programa debido a que no ha sido creado por él en cuestión. Eso lo llevó a cabo la estudiante Carolina Nicolás Martín (y antes Adrián Bargueño) y, por tanto, es ella quien ha elaborado la memoria [7] que explica en detalle el programa. En este sentido y continuando la línea aclaratoria, la parte fundamental de este estudio es la mejora aplicada al simulador. Sin embargo, para el alumno no deja de ser importante exponer en esta memoria, una introducción, estado del arte, metodología y funcionamiento general del simulador para dar un enfoque global de la aplicación y que el lector no tenga necesidad de leerse dos documentos, el de Carolina primero y después éste. Se desea que el lector solamente indague en la memoria de Carolina en busca de profundidad técnica y aclaración sobre fragmentos de código específicos del simulador.

4.1. Descripción del simulador

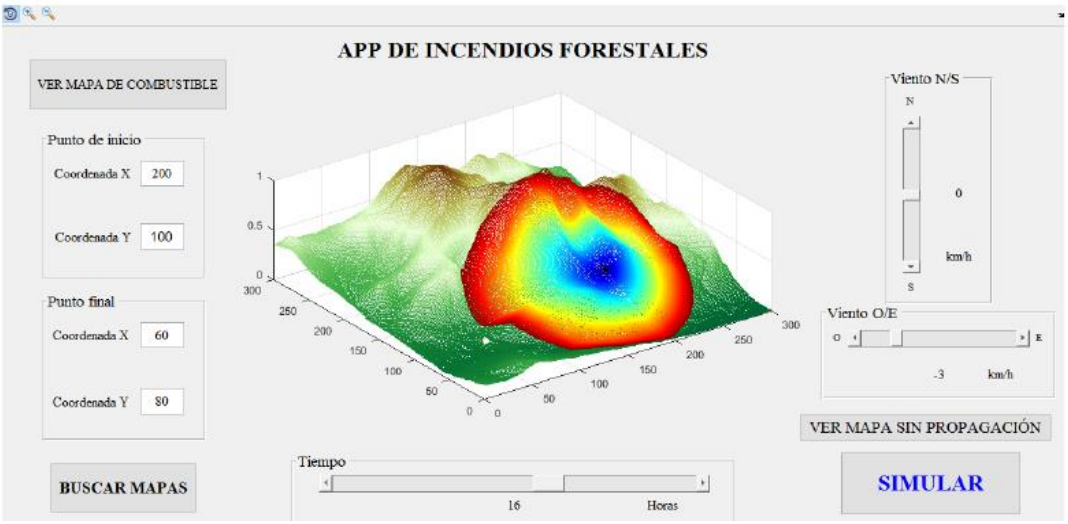


Figura 4.1. Interfaz gráfica de usuario

Esta es la pantalla que el usuario visualiza al encender el programa. Sin necesidad de acudir a otras pantallas se pueden realizar todas las customizaciones de variables excepto la selección del mapa que se carga en otra ventana.

Se procede a describir cada elemento para que el lector sea capaz de establecer sus valores deseados, aunque se puede ver que es intuitivo y sencillo.

- **Selección de mapas:** Cuando se pica el botón de buscar mapa, se accede a otra ventana donde se podrá escoger un archivo previamente guardado, que es la porción de un determinado mapa sobre el que se realizará la propagación. Como especificaciones, el archivo debe ser de tipo .mat y dicho archivo debe contener dos matrices, `elevacion` y `velocidad` y un parámetro binario. La primera matriz hace referencia a la altura de cada punto (coordenada Z) de la porción seleccionada con el fin de poder presentar el mapa en 3D. La segunda es una matriz que representa la capacidad de arder de cada posición expresada en tanto por 1, donde un 0 hace alusión a un terreno que no arde y un 1 a uno que con máxima capacidad de flamabilidad. Por último, contiene una variable interna llamada `cheoshe` que marca el mapa que se ha seleccionado en la simulación, de tal manera que si toma valor 1 el mapa seleccionado es el mapa Cherokee y si toma valor 0 es el Sherman.

  Una restricción para que el simulador funcione debido a cómo se ha diseñado el código base es realizar este paso de selección del mapa el primero. Si se elige otro parámetro antes, el programa deja de funcionar.

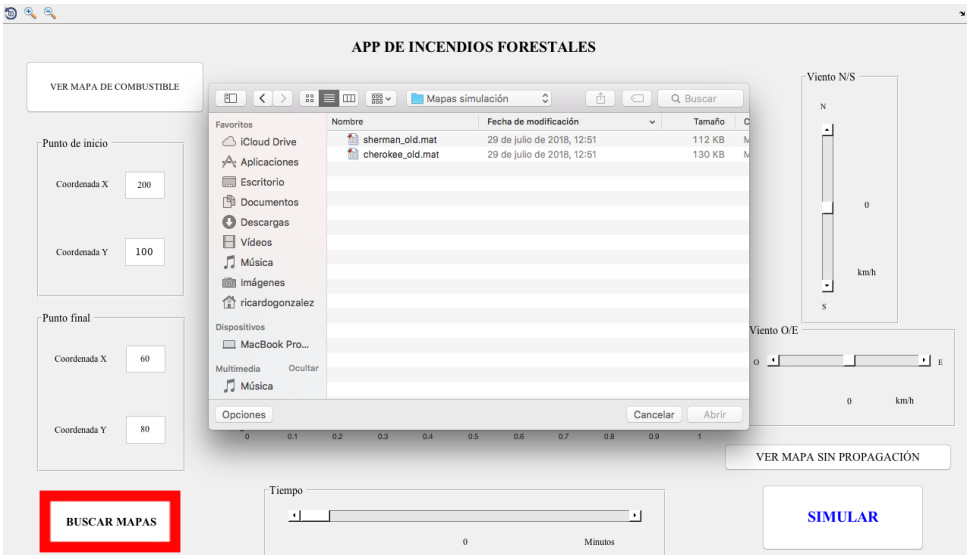


Figura 4.2. Ventana emergente tras presionar el botón “BUSCAR MAPAS”

- **Ver mapa sin propagación:** Inicialmente el cuadro central de la interfaz está en blanco, al picar este botón aparece la porción de mapa seleccionado en 3D antes de llevarse a cabo la simulación.

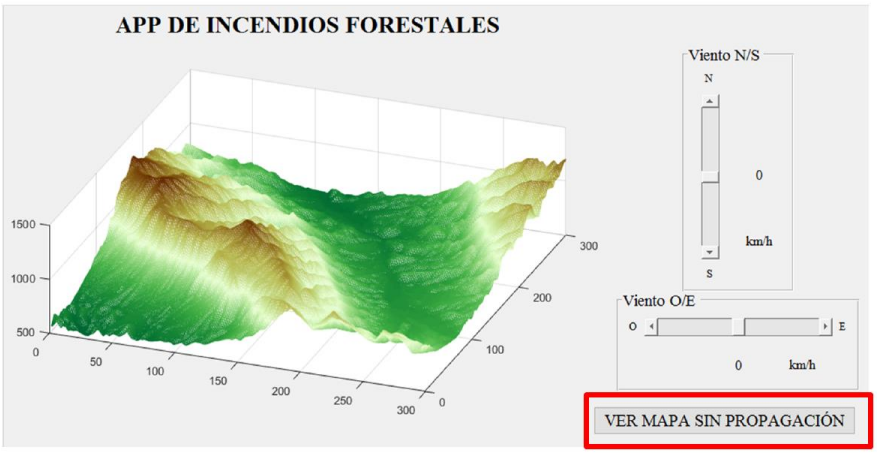


Figura 4.3. Vista del mapa seleccionado en ausencia de la simulación

- **Selección de los puntos iniciales y finales del incendio:** Se permite al usuario elegir el origen posicional del fuego y una posición determinada donde se finalizará la simulación (no implica que el incendio se acabe ahí) que normalmente será un límite físico como el punto de inicio de un pueblo.

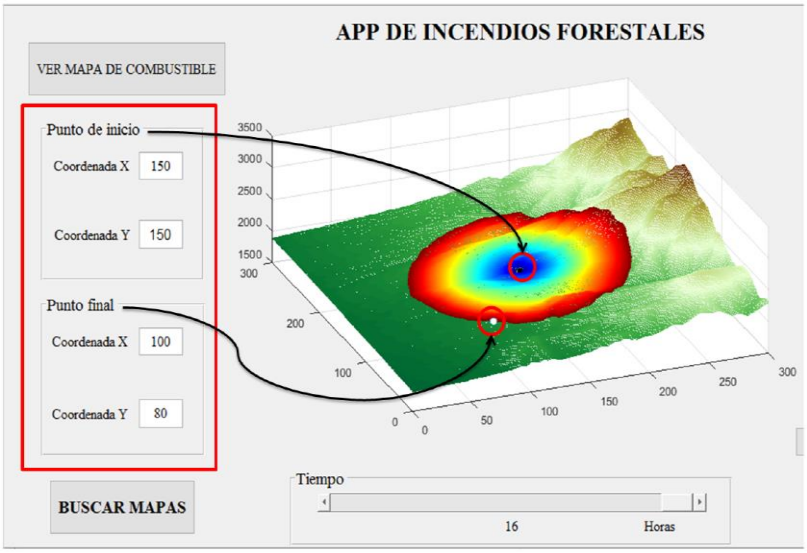


Figura 4.4. Ejemplo de simulación con inicio en el punto negro y fin en el blanco

- **Personalización del viento:** En los dos *sliders* perpendiculares de la parte derecha se permiten establecer la cantidad del viento y su dirección de avance para la simulación. Sus posibles valores van de 5 a -5. La negatividad del valor tiene que ver con el sentido de avance del viento.

Se propone un ejemplo empleando la figura 4.5. para entender el funcionamiento de estos dos elementos complementarios:

Un valor de 4 positivo en el *slider* horizontal y un valor de 0 en el vertical como el de la figura 4.5. implica que el viento que sopla lo hace hacia el Este (viento del Oeste) y avanza a 4 km/h. Pero si fuera ese mismo valor, pero negativo, el viento avanzaría hacia el Oeste (viento del Este) con esa misma velocidad, 4 km/h.

Un matiz lingüístico importante es que, al nombrar un viento con su dirección, hace referencia al lugar de dónde procede y no hacia dónde va. Por lo que, de aquí en adelante, cuando se haga alusión al viento del Norte, se refiere al viento con sentido Norte – Sur. Como expresión alternativa se empleará para designar el viento Norte la frase “viento hacia el Sur”.

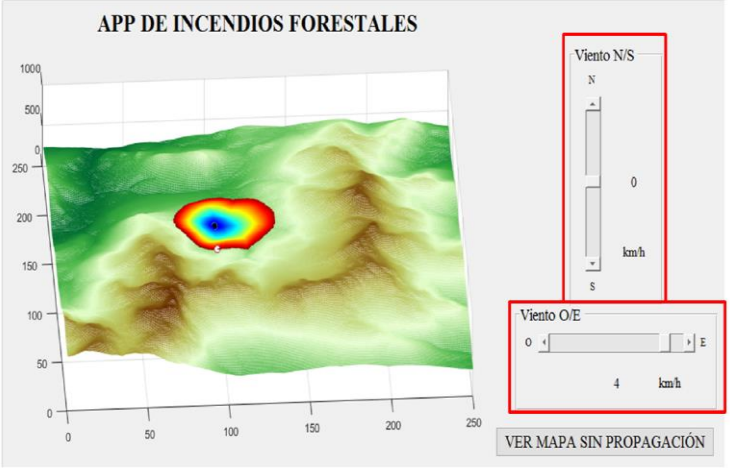


Figura 4.5. Ajuste del parámetro viento

Se presenta en la figura 4.6. otro caso visual de la importancia del viento en la propagación del incendio. Se puede observar cómo en función de hacia dónde sopla el viento, el avance del incendio es más notable.

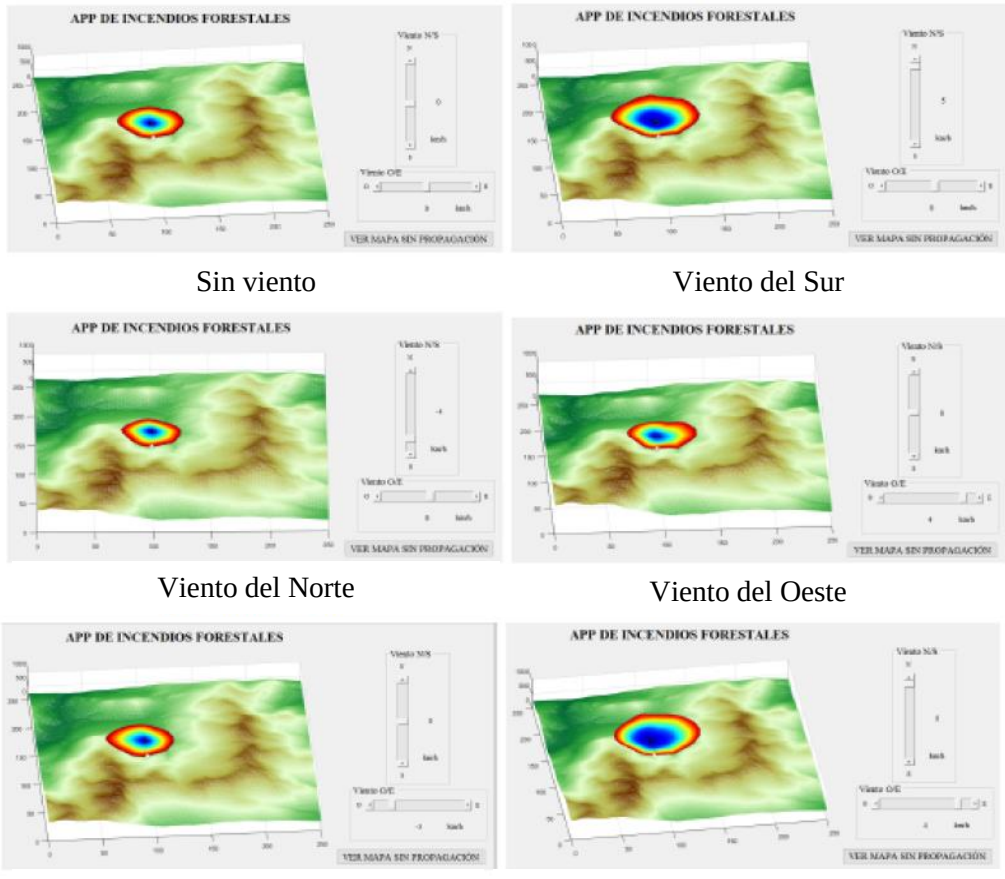


Figura 4.6. Visualización de la diferencia de propagación del fuego según cómo sople el viento

- **Elección del tiempo de expansión del incendio:** Para las posiciones de inicio y fin del fuego, el simulador tiene la funcionalidad de visualizar la simulación en posiciones intermedias. El *slider* de la parte inferior de la interfaz se ajusta con un tiempo máximo personalizado según los valores establecidos del viento y posiciones de inicio y fin. Una vez ajustado el tiempo máximo, se pueden obtener momentos intermedios según se suelte la barra del *slider*.

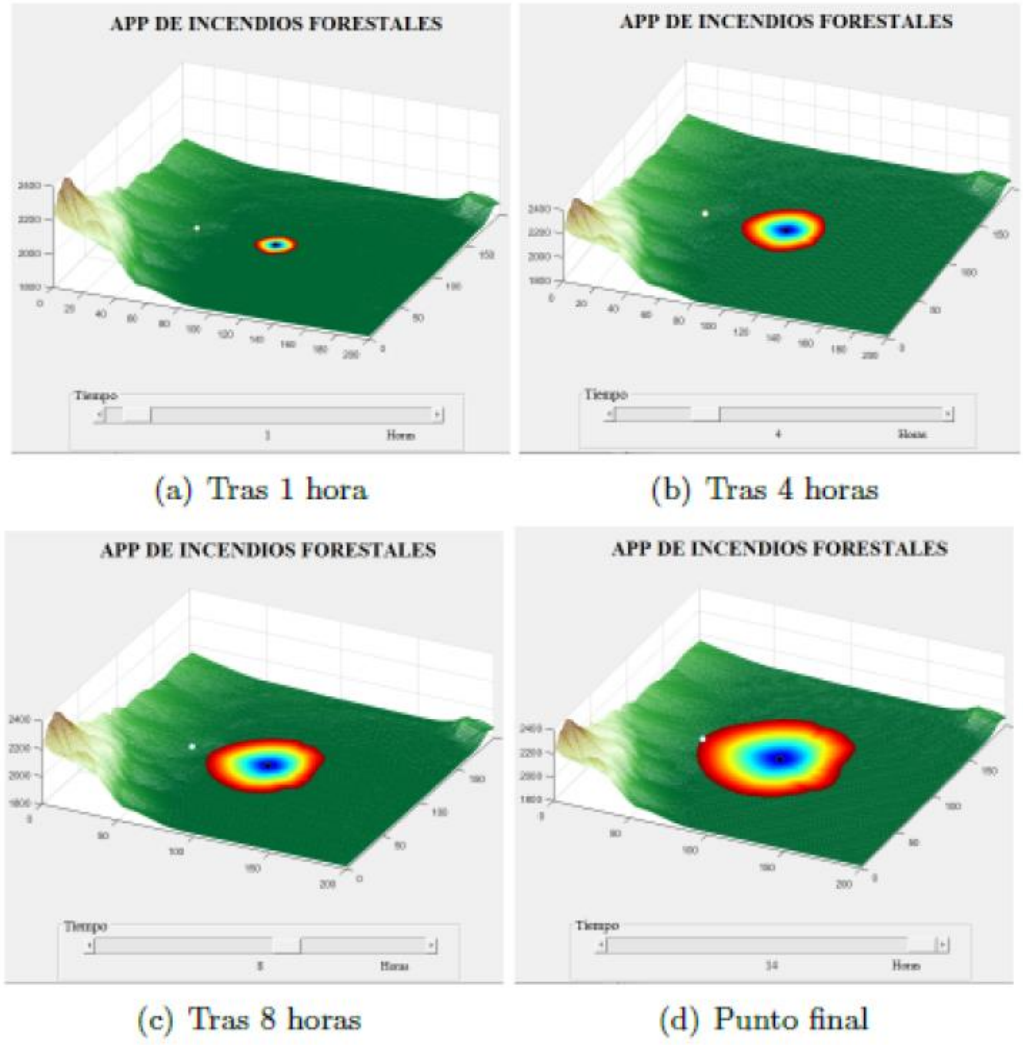


Figura 4.7. Visualización de momentos intermedios de la simulación

- **Observación del mapa de flamabilidad:** Si se presiona el botón en la zona superior izquierda "VER MAPA DE COMBUSTIBLE" aparece una ventana que describe la capacidad de arder del área de mapa seleccionado con la misma posición de los ejes para que el usuario tenga más información del terreno y pueda elegir mejor los valores de los parámetros para conseguir una simulación más valiosa. Este mapa suele ser usado para tomar en consideración la expansión de la onda. Por tanto, su visualización se produce antes que la selección de los valores de los parámetros, como el viento.

Se presentan las figuras 4.8. y 4.9. con el propósito de mostrar al lector lo explicado previamente. Es coherente que la propagación sea mayor hacia la montaña. Por 2 motivos. El primero es debido a la inclinación del terreno, que, como ya se ha explicado, el fuego tiende a avanzar más hacia zonas altas que bajas o rectas. El segundo es la combustibilidad específica del terreno. Como se observa en la figura 4.9. en la parte izquierda hay una zona rectangular de mínima capacidad de arder, probablemente se trate de un lago u otra acumulación de agua. Es obvio que en esa dirección el avance debe ser significativamente menor. Esto mismo queda demostrado en la figura 4.8. observando la menor expansión del fuego en el lado izquierdo.

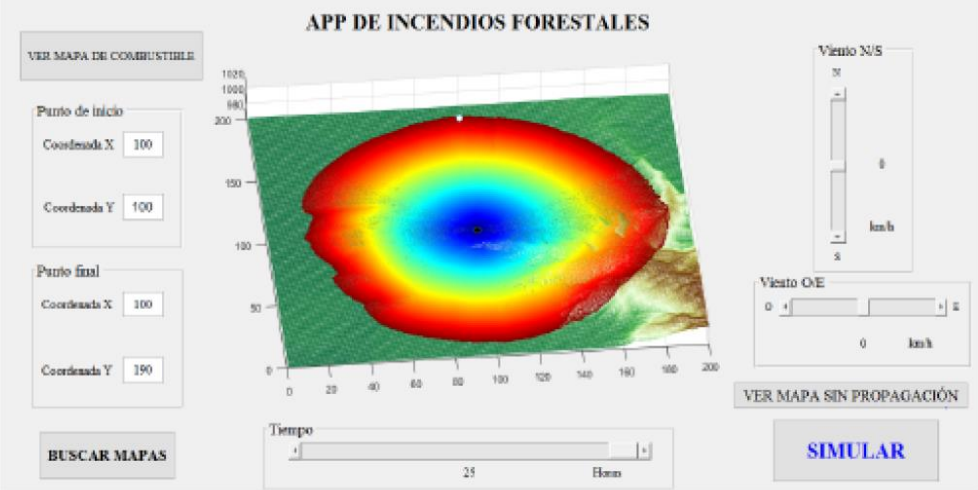


Figura 4.8. Ejemplo de simulación

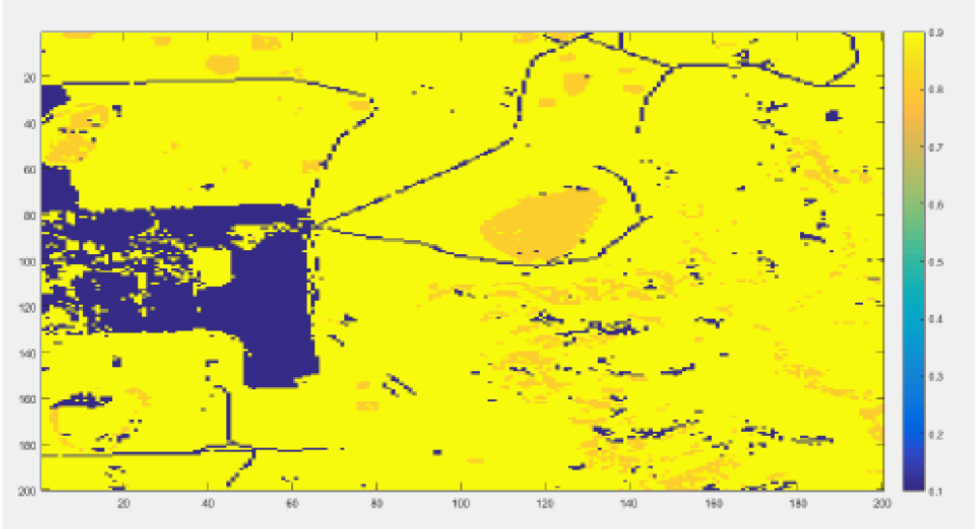


Figura 4.9. Mapa de combustible de la figura 4.8.

- **Simulación:** En el botón "SIMULAR" acaba la interacción del usuario con el programa. Es el botón que hace funcionar el simulador. Siempre que se modifique el valor de alguna variable debe pulsarse este botón para aplicar a la simulación dicha modificación.

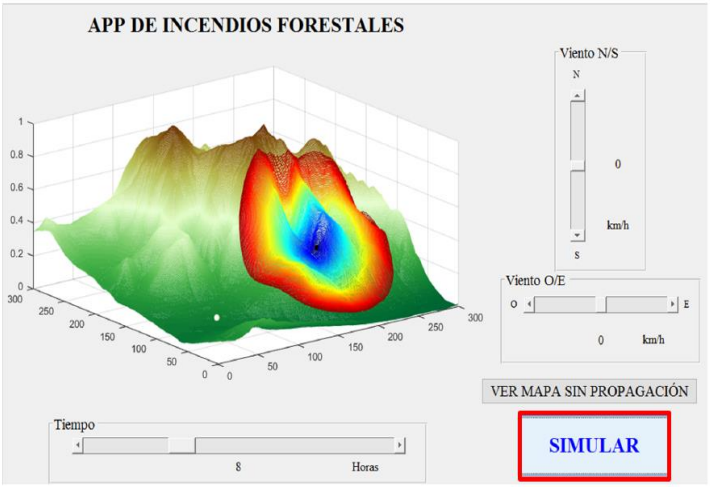


Figura 4.10. Situación del botón Simular en la interfaz de usuario

4.2. Funcionamiento general del simulador

Se admite que el simulador no destaca por su complejidad técnica con funciones recurrentes y relaciones complicadas, de hecho, se ha realizado el programa siendo consciente de ello, e incluso obligándose a ello pues la sencillez y claridad es una premisa fundamental en el simulador. Su construcción se puede resumir en la programación de una interfaz gráfica de usuario de MatLab, o también conocido como GUI. Sin embargo, no es menos importante la manera en la que los *inputs* llegan al algoritmo FM y la manera en la que son tratados los datos resultantes del algoritmo para conseguir una visualización agradable y efectiva del usuario. De forma poco convencional, se ha creado una función que contiene a ambos procesos y, por tanto, adquiere un peso comparable al GUI. Esta función se llama `ejecuta_FM_prueba_R` y fue creada por Carolina y modificada por el alumno de este TFG. Se muestran sus líneas íntegras en el apéndice C apartado C.3. Pero se explican sus puntos clave a continuación:

Esta función recibe como parámetros los 3 archivos que contiene el mapa, las matrices `elevacion` y `velocidad` (archivos .map), y la variable binaria `cheoshe`. Recibe también los puntos de inicio y fin, los valores del viento y el límite de tiempo junto con el tiempo intermedio (en caso de que lo haya) establecido en el correspondiente *slider*.

En esta función encontramos la llamada al algoritmo Fast Marching. En realidad, se llama a una función codificada de FM en C++ usada debido a su mayor velocidad de ejecución en C++ que en MatLab, `fast_marching_vectorial_2d`. Fue creada por Gabriel Peyre y modificada por Santiago Garrido. Su *output* es el mapa de distancias D anteriormente estudiado junto con una matriz del estado de todas las posiciones recorridas llamada S, en especial, las posiciones vacías o muertas. Todo ello para un origen y final de coordenadas, un tope de repeticiones, la matriz `velocidad` y campos vectoriales, uno para el eje X y otro para el Y.

En resumen, dentro de esta función, se trata los parámetros de entrada de `fast_marching_vectorial_2d` y se prepara el mapa de distancias D para la visualización de la expansión del incendio sobre el mapa seleccionado en 3D.

En las líneas de texto previas, se ha comentado el tratamiento de los parámetros de entrada de Fast Marching. Dicho tratamiento consiste en el cambio o adaptación de los datos que llegan de los valores seleccionados por el usuario para que lleguen al algoritmo de la manera apropiada. Entre los más destacados, un cambio es que la matriz `velocidad` debe ser reorientada ya que el algoritmo lo recibe en unos ejes distintos a los que genera el usuario. Para ello, se usa la función `flip_vertical` y el recurso de la matriz traspuesta. Otro caso, es el cambio de estructura para almacenar los valores de viento. Se reciben como variables simples que almacenan un valor entre -5 y 5, pero en FM tienen que aparecer como matrices cuadradas de no más de 400 filas a ser posible (MatLab falla en caso de matrices grandes). La solución es multiplicar esa variable simple por una matriz de unos de las dimensiones requeridas y así se obtiene una matriz con todas las posiciones de igual valor. De tal forma que se obtienen 2 vectores de viento, `VX` y `VY`. La última adaptación representativa, es la integración de la inclinación del terreno como *input*. A la matriz `elevacion`, se le aplica el cálculo del gradiente para contabilizar dicha pendiente y se almacena en dos matrices de igual dimensión que `elevacion` con los gradientes del eje X y del eje Y, `DX` y `DY`. Conjuntamente, la suma proporcional de `VX` y `VY` con `DX` y `DY` da lugar a las dos matrices que son establecidas como parámetros de FM en su llamada, `FX` y `FY`.

Igualmente ocurre con el *output* de `fast_marching_vectorial_2d`, éste debe ser tratado pues el mapa de distancias D no está adaptado para su visualización esperada. Algunas de las adaptaciones más relevantes de esta parte del código se detallan a continuación:

- El mapa de distancias `D` debe ser reorientada, al igual que lo fue la matriz `elevacion` para ser *input* de FM. De hecho, es consecuencia de la primera reorientación de `elevacion`, y lo que se busca es revertir el orden de filas y columnas otra vez. De la misma manera que se hizo con la matriz `elevacion`, pero a la inversa.

- Puesto que la propagación se visualiza sobre el mapa original (el cual se representa mediante la llamada de la función `mesh` aplicada a la matriz `elevacion` y después, asignándole un `colormap` adecuado), como un

conjunto de círculos de distinto color, es necesario establecer un número de líneas de contorno (todas juntas formarán esa superficie circular compuesta por varios anillos concéntricos seguidos de distintos colores), los puntos que conforman cada nivel con sus respectivas coordenadas y las alturas de cada punto. Para ello se emplea la función `contour` que genera una matriz C capaz de almacenar todos estos valores de forma ordenada y eficaz.

- Se desarrollan dos bucles `while` para llevar a crear la vista de los contornos y asignar colores a cada línea de contorno.

Este apartado ha sido redactado a alto nivel para alinearse con el resto del trabajo en la búsqueda de lograr que el lector adquiera una perspectiva global del programa. Es por esto por lo que se han nombrado las adaptaciones, funciones y bucles más relevantes del código. Sin embargo, este contenido no es parte esencial de este trabajo pues pertenece al desarrollo realizado por la alumna que creó el programa, Carolina. De ahí, que no se profundice en dichos elementos. Pese a ello, el lector puede encontrar el detalle técnico en el documento propicio, la memoria del TFG de Carolina [7], donde de forma excelente, queda especificado el funcionamiento de cada elemento nombrado aquí. En especial, se puede encontrar en el apartado 4.3. de dicho documento.

4.3. Implementación de la mejora del simulador

El núcleo de este proyecto de mejora que el alumno a elegido para realizar su Trabajo de Fin de Grado se explica en este capítulo. Se trata pues, de la parte más relevante del documento y se redactará en torno a una estructura perfectamente definida que puede apreciarse en la figura 4.11.

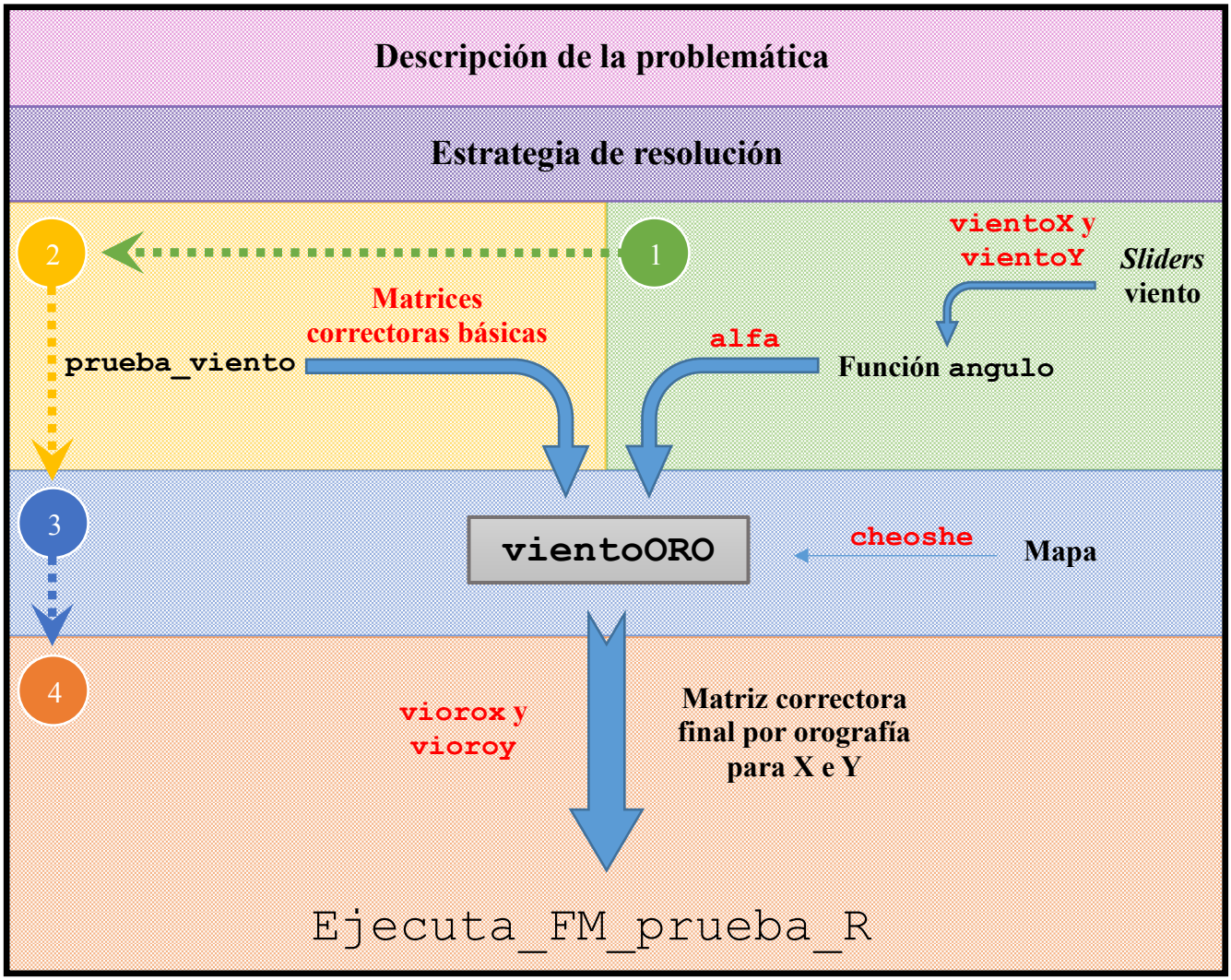


Figura 4.11. Estructura del apartado 4.3.

4.3.1. Descripción de la problemática

El efecto Föhn según *Wikipedia* [26], "se produce en relieves montañosos cuando una masa de aire cálido y húmedo es forzada a ascender para salvar ese obstáculo, lo que provoca que el vapor de aire se enfríe y sufra un proceso de sublimación precipitándose en las laderas de barlovento donde se forman nubes y lluvias orográficas."

Este fenómeno evidencia la influencia que puede llegar a tener indirectamente la orografía (en este caso una montaña) en la aparición de fenómenos medioambientales, debido a que directamente influye sobre el avance del viento. En el caso del efecto Föhn, en el que el viento en lugar de seguir el sentido que mantenía antes de encontrarse con el obstáculo es obligado a reducir y adaptar tanto su velocidad como su orientación, que provoca dicho fenómeno.

Es cierto que el efecto Föhn está referido a grandes masas de aire y a obstáculos enormes como una montaña. Sin embargo, en una escala visiblemente menor se produce el mismo evento: El viento también es forzado a modificar su trayectoria cuando se presentan en su camino elementos más pequeños como árboles, rocas, valles, y otras formas de relieve. Lo que implica que este factor de la orografía tiene presencia en todos los puntos de una superficie y debe ser teniendo en cuenta como tal.

Sucede lo mismo con otros fenómenos naturales. La orografía tiene un gran impacto en ellos, sobre todo, en aquellos en los que el viento juega un papel fundamental. Que es el caso de los incendios forestales, donde el viento suele actuar como "catalizador" de un fuego forestal.

Se busca con esta explicación, visibilizar el impacto de la orografía en los incendios. Que previamente no se ha tenido en cuenta en el desarrollo del simulador pero que conforma la mejora por la que se realiza este TFG.

Profundizando en el entendimiento de dicha problemática, mediante esta mejora, el algoritmo debe plasmar el hecho de que un mismo obstáculo natural, por pequeño que sea, no influye de la misma manera en el avance del viento, si éste sopla en una dirección o en otra. Con esta premisa en mente, el algoritmo no sólo debe contar con la influencia propia de la orografía, sino que debe conseguir adjudicar la influencia precisa que corresponde a cada sentido de avance del viento. Para cada una de las 360 posibles orientaciones del viento debe establecerse un impacto real de la tipología del terreno en el viento. Permítase redondear a 360 las opciones de dirección del viento (una por cada grado sexagesimal) como método de simplificación de cálculo en base a la obtención de demostraciones reales y valiosas, pues pesa más que la posible pérdida resolutiva tras realizar esta estimación.

4.3.2. Estrategia de resolución

Tras definir y delimitar el objetivo a conseguir, se deben valorar las mejores alternativas para implementar una solución. La opción de almacenar 360 matrices y seleccionar la necesaria en cada simulación va en contra de uno de los pilares básicos de este proyecto. Austeridad técnica. Esto no es viable debido a su alto coste computacional en relación con la necesidad que cubre dentro del programa. La solución que mejor se adapta a los requerimientos se basa en trigonometría.

Se procede a describir el proceso resolutivo como iniciación en el entendimiento de éste, pero sin olvidar que es explicado en mayor profundidad en los apartados venideros. La acción principal es la obtención de unas matrices cuyos valores contabilizan el impacto de la orografía en el viento antes de realizar la simulación. El ahorro computacional aparece en el hecho de almacenar sólo 4 de 360 posibles matrices que plasmarían este impacto para cada posible orientación del viento en la simulación. Se comprende el ahorro realizado. Pero lógicamente es necesario generar una estructura que, a partir de estas 4 matrices, sea capaz de obtener una determinada que no esté entre las almacenadas. Se ha decidido que las 4 matrices previamente generadas sean aquellas que hagan referencia al viento del Norte, Sur, Este y Oeste puros. Por tanto, quedarían cubiertas las matrices necesarias cuando en la simulación el usuario elija un viento con orientación de 0º, 180º, 270º y 90º. Sin embargo, quedarían 355 casos en los que no se posee la matriz específica que contiene el impacto del terreno en el viento para esa orientación del viento. No es casualidad que se elijan 4 orientaciones de vientos que son perpendiculares entre sí y paralelas a los ejes de coordenadas. Se han seleccionado esas 4 específicamente para poder aplicar relaciones trigonométricas y poder obtener el resto de las matrices correctoras.

Comenzada la simulación, teniendo las 4 matrices correctoras básicas ya almacenadas, se necesita hallar el valor del ángulo establecido por el usuario al elegir los valores de velocidad y dirección del viento por medio de los 2 *sliders* situados en la parte derecha de la interfaz gráfica. Como el usuario no escribe el ángulo directamente, es necesario obtenerlo internamente. Esto se llevará a cabo por medio de la función `angulo` haciendo uso una vez más de la trigonometría para calcularlo.

Con estos *inputs* ya calculados, se planea diseñar una función que usando senos y cosenos, a través del ángulo elegido, rellene la matriz correctora final para ese ángulo. Esta función será `vientoORO` y se puede considerar como la función coordinadora de la mejora. Como ejemplo, dentro de ella, se podría dar un caso en el que se busque simular con viento del Noroeste, entonces deberá formar la nueva y definitiva matriz correctora a partir de las matrices correctoras básicas del viento del Norte y del viento del Oeste, que ya han sido obtenidas previamente. Haciendo uso de la trigonometría a partir del ángulo implícito generado al seleccionar un viento del Noroeste en la pantalla de usuario.

A partir de este punto, se explica el proceso de implementación de la mejora del simulador siguiendo la estructura de la figura 4.11. en los próximos 4 apartados.

4.3.3. Función `angulo`

Como primera tarea operativa se debe diseñar una manera de obtener una variable simple con el valor del ángulo (con límites de 0 a 360) establecido por el usuario al calibrar los parámetros del viento a través de los dos *sliders* contenidos en la zona derecha de la interfaz gráfica.

La opción elegida es la creación de una función que recibe como parámetros los valores de ambos *sliders* y expulsa como *output* una variable de tipo *double* con un valor entre 0 y 360 redondeado a las unidades que representa el ángulo con el que sopla el viento en cada simulación. El código completo puede ser observado en el apéndice C, apartado C.3.

Primeramente, es necesario establecer el sistema de referencia para medir el ángulo del viento. Se ha decidido establecer el origen en el punto más alto de la circunferencia que designa la orientación del viento hacia el Norte puro (sentido Sur – Norte). De tal forma que, tal y como muestra la figura 4.12., cuando haya viento hacia el Oeste, su ángulo será 270°, para viento hacia el Norte 180° y para viento hacia el Este 90°.

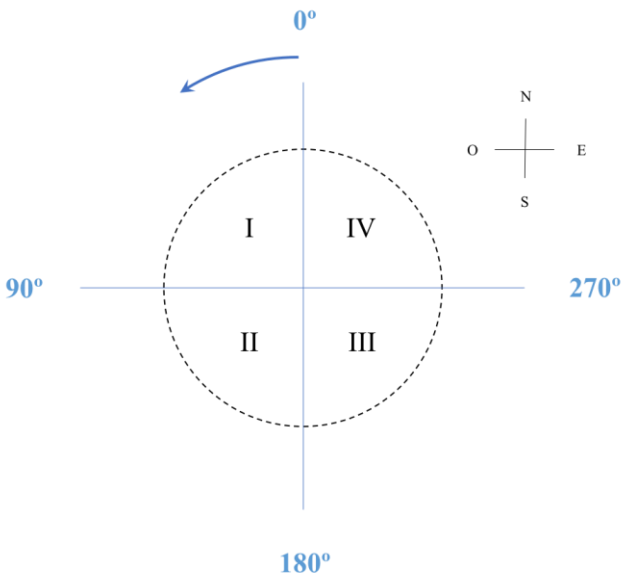


Figura 4.12. Representación del sistema de medición de la orientación del viento del simulador en base a los puntos cardinales.

Tras establecer el sistema de referencia, se asocian las variables del viento a los ejes X e Y de la siguiente manera:

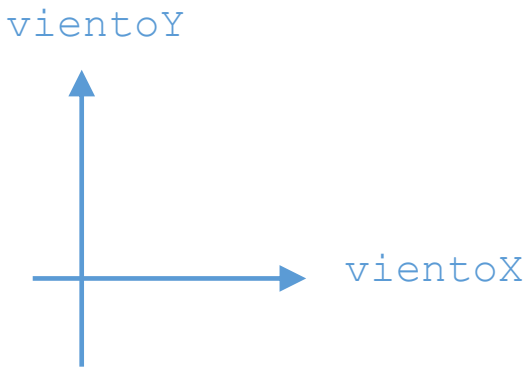


Figura 4.13. Asignación de los ejes de referencia

Después de elegir las referencias, se debe explicar para cada cuadrante el procedimiento matemático que calcula el valor del ángulo a partir de `vientoX` y `vientoY`, variables que almacenan la cantidad de viento y su dirección establecidos por el usuario en los *sliders* de viento N/S y E/O.

4.3.3.1. Función `angulo` para cuadrante I

```
%Cuadrante I
if((vientoX<0) && (vientoY>0))

%atand es la función del arcotangente en MatLab.
    alfa=atand(-vientoX / vientoY);
```

Para la explicación correspondiente, se presenta al lector la figura 4.14.

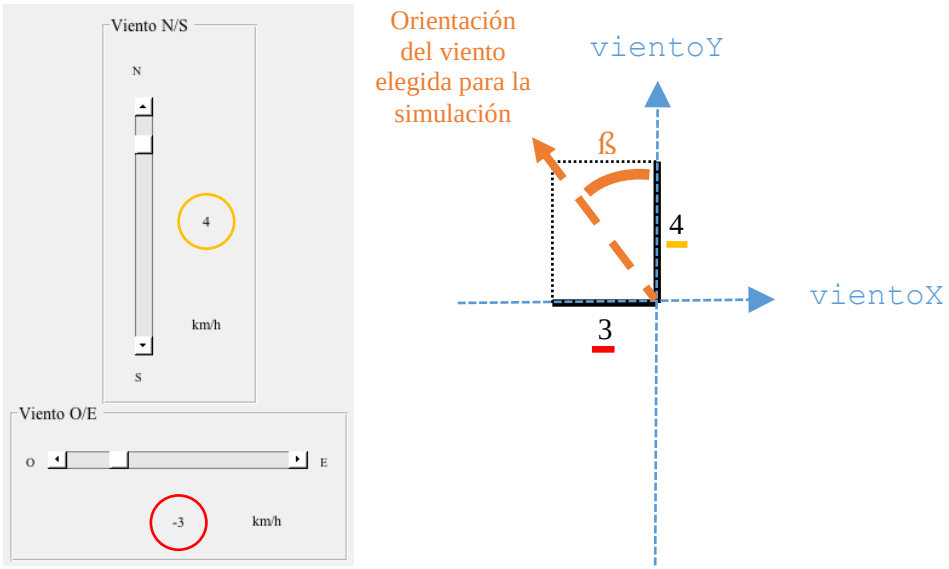


Figura 4.14. Ejemplo de viento con ángulo del cuadrante I

Aplicando la relación trigonométrica de la tangente a los valores de la figura 4.14. se tiene que:

$$\tan ß = \frac{|vientoX|}{vientoY} \tag{4.1}$$

De la fórmula (4.1) se obtiene que:

$$ß = \tan^{-1} \frac{|vientoX|}{vientoY} \tag{4.2}$$

Donde si se sustituye por los valores numéricos, se obtiene el valor de ß:

$$ß = \tan^{-1} \frac{3}{4} = 36{,}87° \tag{4.3}$$

Por último, se asigna el valor de ß a la variable `alfa` ya que en el caso del cuadrante I el ángulo coincide con ß:

$$\texttt{alfa} = ß = 36{,}87° \tag{4.4}$$

Que redondeado a las unidades se termina con

$$alfa = 37° \tag{4.5}$$

4.3.3.2. Función `angulo` para cuadrante II

```
%Cuadrante II
elseif((vientoX <0) &&( vientoY <0))

%atand es la función arcotangente en MatLab
    alfa=180-atand(-vientoX /- vientoY);
```

Para la explicación correspondiente, se presenta al lector la figura 4.15.

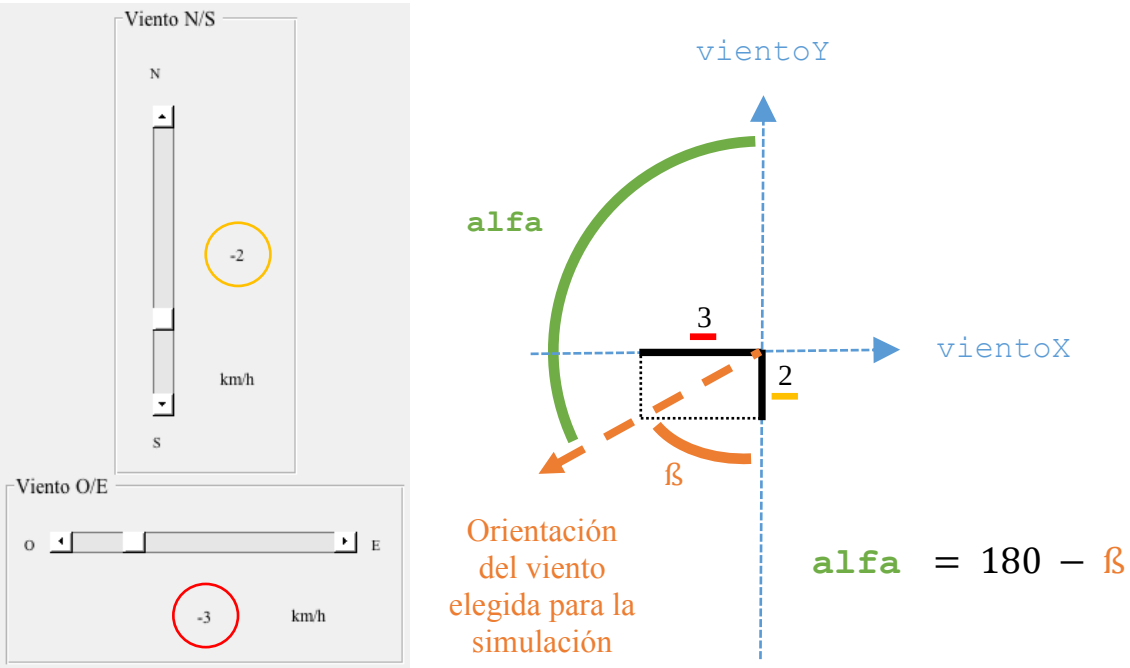


Figura 4.15. Ejemplo de viento con ángulo del cuadrante II

Aplicando la relación trigonométrica de la tangente a los valores de la figura 4.15. se tiene que:

$$\tan ß = \frac{|vientoX|}{|vientoY|} \tag{4.6}$$

De la fórmula (4.6) se obtiene que:

$$ß = \tan^{-1} \frac{|vientoX|}{|vientoY|} \tag{4.7}$$

Donde si se sustituye por los valores numéricos, se obtiene el valor de ß:

$$ß = \tan^{-1} \frac{3}{2} = 56{,}31° \tag{4.8}$$

Por último, se calcula el valor de `alfa` a partir de:

$$\texttt{alfa} = 180 - ß \tag{4.9}$$

Y se obtiene que:

$$\texttt{alfa} = 180 - ß = 143{,}13° \tag{4.10}$$

Que redondeado a las unidades se termina con

$$\texttt{alfa} = 143° \tag{4.11}$$

4.3.3.3. Función `angulo` para cuadrante III

```
%Cuadrante III
elseif ((vientoX >0) && (vientoY <0))

%atand es la función arcotangente en MatLab
    alfa=180+atand(vientoX /- vientoY);
```

Para la explicación correspondiente, se presenta al lector la figura 4.16.

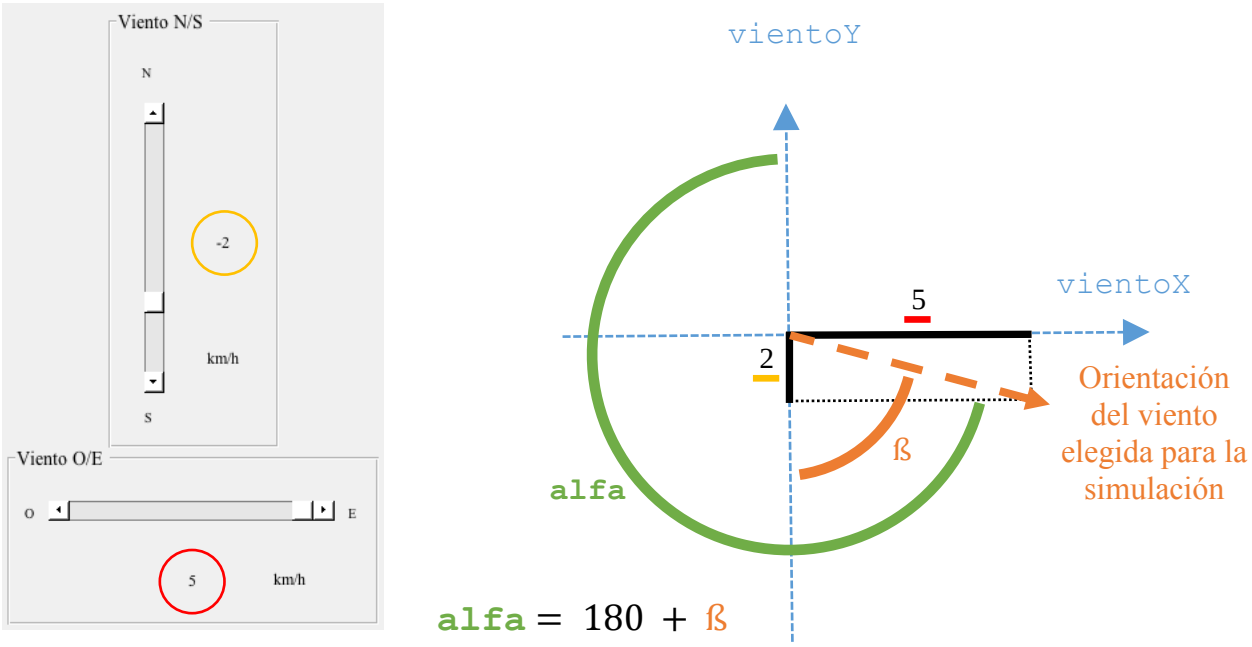


Figura 4.16. Ejemplo de viento con ángulo del cuadrante III

Aplicando la relación trigonométrica de la tangente a los valores de la figura 4.16. se tiene que:

$$\tan ß = \frac{vientoX}{|vientoY|} \tag{4.12}$$

De la fórmula (4.12) se obtiene que:

$$ß = \tan^{-1}\frac{vientoX}{|vientoY|} \tag{4.13}$$

Donde si se sustituye por los valores numéricos, se obtiene el valor de ß:

$$ß = \tan^{-1}\frac{5}{2} = 68{,}20° \tag{4.14}$$

Por último, se calcula el valor de `alfa` a partir de:

$$\texttt{alfa} = 180 + ß \tag{4.15}$$

Y se obtiene que

$$\texttt{alfa} = 180 + ß = 248{,}20° \tag{4.16}$$

Que redondeado a las unidades se termina con

$$\texttt{alfa} = 248° \tag{4.17}$$

4.3.3.4. Función `angulo` para cuadrante IV

```
%Cuadrante IV
elseif ((vientoX >0) && (vientoY >0))

%atand es la función arcotangente de MatLab

    alfa=360-atand(vientoX / vientoY);
```

Para la explicación que procede, se presenta al lector la figura 4.17.

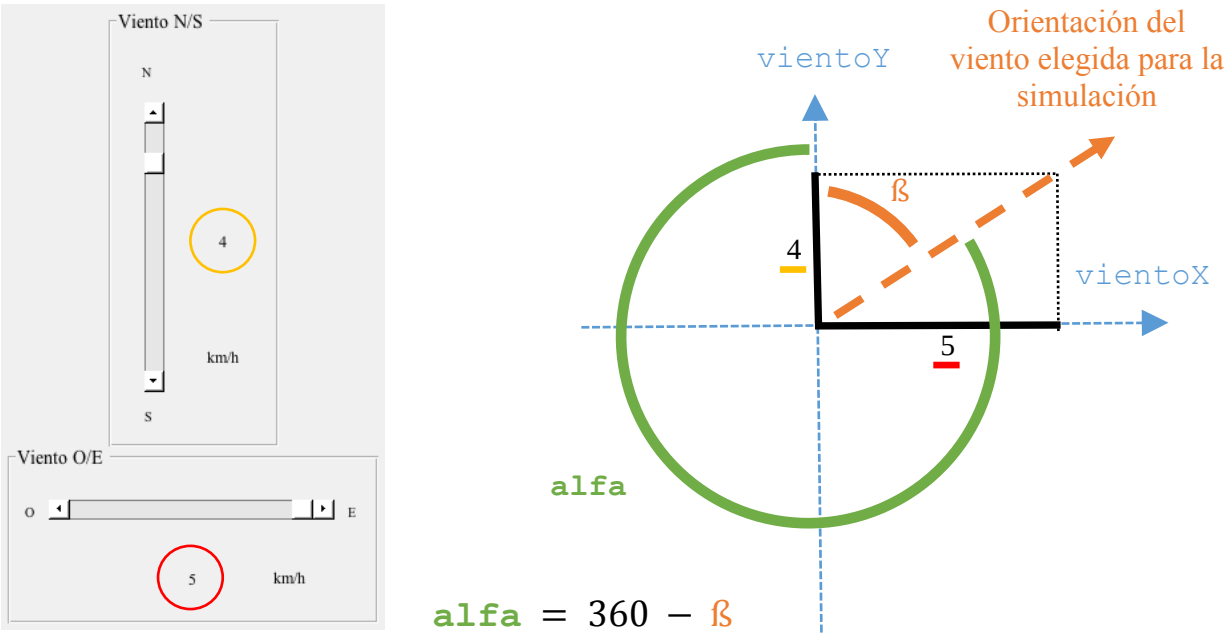


Figura 4.17. Ejemplo de viento con ángulo del cuadrante IV

Aplicando la relación trigonométrica de la tangente a los valores de la figura 4.17. se tiene que:

$$\tan ß = \frac{vientoX}{vientoY} \tag{4.18}$$

De la fórmula (4.18) se obtiene que:

$$ß = \tan^{-1} \frac{vientoX}{vientoY} \tag{4.19}$$

Donde si se sustituye por los valores numéricos, se obtiene el valor de ß:

$$ß = \tan^{-1} \frac{5}{4} = 51{,}34° \tag{4.20}$$

Por último, se calcula el valor de `alfa` a partir de:

$$\texttt{alfa} = 360 - ß \tag{4.21}$$

Y se obtiene que

$$\texttt{alfa} = 360 - ß = 308{,}66° \tag{4.22}$$

Que redondeado a las unidades se termina con

$$\texttt{alfa} = 309° \tag{4.23}$$

4.3.3.5. Función `angulo` para los extremos

En este breve apartado, el alumno aprovecha para puntualizar que en la función `angulo` los límites entre cuadrantes son introducidos en bucles `if` individuales en los que se le asigna manualmente el valor del ángulo. De tal forma que cuando se desea simular con viento hacia el Norte (moviendo solamente la barra del *slider* de viento N/S hacia valores positivos) a la variable `alfa` se le asigna el valor 0. Lo mismo para el viento hacia el Sur que se le asigna el valor 180, hacia el Este con asignación de 270 y el viento hacia el Oeste donde a la variable `alfa` se le asigna el valor entero 90.

```
%Viento hacia el Norte
elseif((vientoX ==0) && (vientoY >0))

    alfa=0;

%Viento hacia el Sur
elseif((vientoX ==0) && (vientoY <0))
```

```
    alfa=180;

%Viento hacia el Oeste
elseif((vientoX <0) && (vientoY ==0))

    alfa=90;

%Viento hacia el Este
elseif((vientoX >0) && (vientoY ==0))

    alfa=270;
```

4.3.4. Función `prueba_viento`

Como se ha comentado en el planteamiento del método resolutivo, esta función se ejecuta antes de la primera simulación ya que su tarea es generar las 4 matrices correctoras básicas que plasman el impacto de la orografía en el comportamiento del viento. Es por esto por lo que es considerada la base de la mejora implementada en el simulador, además de que es la que extrae el valor generado por el algoritmo Fast Marching. Pese a que el funcionamiento de FM ha sido detalladamente explicado a nivel teórico en el capítulo 3, es valioso comentar a alto nivel su aplicación y su relación con las variables:

- FM simula la expansión de una onda, pudiendo captar tanto el camino que sigue como el tiempo que dura la onda en su propagación, además de la forma que dibuja el avance de la onda.

- Los parámetros de entrada que precisa FM son los siguientes:

  - Mapa de grises en 3D de la sección de terreno donde se realizará la simulación. Un mapa de grises es una matriz tridimensional cuyas posiciones pueden tomar sólo 2 valores: 0 y 1.

  - Matriz de almacenamiento de coordenadas de los puntos desde los que se pretende realizar la expansión de onda.

  - Paquete de opciones que hacen referencia al número máximo de iteraciones en la simulación y el punto donde acaba ésta.

- El *output* de FM es la matriz `D`, que es un mapa de distancias de la expansión en cada posición de la matriz de los puntos de inicio. Con esta matriz resultante se establece la evolución de la expansión, que es el suceso clave de esta función, por lo que la matriz `D` es un elemento fundamental del proyecto. Además, se genera

otra matriz que notifica el estado final de todos los puntos del mapa de grises tras la simulación. Cuáles son los puntos muertos, vecinos y desconocidos.

Fast Marching se considera ideal para ser aplicado en los cálculos necesarios del simulador porque los resultados de la expansión de una onda guardan relación con la expansión de un incendio. Pero, centrándose en la función `prueba_viento` en concreto, que calcula el impacto de la orografía en el viento, ofrece la posibilidad de obtener una aproximación del comportamiento de un fluido durante su trascurso al enfrentar cualquier tipo de obstáculos por medio de producir expansiones de onda en posiciones estratégicamente colocadas.

De tal forma que, como ejemplo, si se deseara conocer la manera en la que el agua de un río sortea las piedras, los troncos de los árboles y otros objetos, se podría conseguir una simulación certera de dicho avance usando el algoritmo de Fast Marching. El modo de hacerlo sería simular primero la expansión de onda correspondiente al agua que baja por el río. En este punto, las líneas de avance serían paralelas y constantes. Para introducir dichos obstáculos en el avance, es necesario modificar esas líneas de avance de la manera que mejor presente a cada obstáculo. La mejor opción es lanzar expansiones de ondas que tengan como puntos de inicio todo el contorno de cada obstáculo de tal forma que las líneas de flujo de la primera onda (el agua) se verán alteradas por líneas de flujo provenientes en sentido contrario desde los distintos elementos que obstaculizan el agua y por tanto en esos puntos, las líneas de avance del agua del río deberán abrirse hacia los lados y el valor de avance de cada punto del agua que tenga delante una obstáculo (una roca) será reducido ya que el agua que se encuentra con una roca frena su velocidad y busca propagarse por aquellos caminos que pueda mantener su curso, que normalmente es pasando por los lados del obstáculo.

Cabe recordar que la función `prueba_viento` es ejecutada antes de la primera simulación y que su valor es la producción de esas matrices correctoras de valores fijos a las que recurre el programa para plasmar el impacto de la orografía en el viento. Por otro lado, dicha función realiza 2 llamadas a FM, por lo que hace 2 series de expansiones de onda para obtener la matriz `D` que busca.

Una simula al viento meteorológico que cruza de manera uniforme vastas extensiones de terreno (esta serie de expansiones comienza desde una de las caras laterales del cubo matricial de la sección del terreno utilizado) y la otra plasma la influencia de la orografía en el viento (esta serie de expansiones comienza en una de las 4 caras laterales del mapa de grises dependiendo del viento que se esté ejecutando, viento hacia el Norte, Sur, Este u Oeste). De tal manera que el resultado de ambas expansiones es la matriz `D` obtenida como *output* de la segunda llamada a Fast Marching. Como matiz aclaratorio, aunque el resultado final es el mismo, el código escrito realiza las expansiones de onda al contrario que lo explicado, primero la correspondiente al viento meteorológico y después la que corresponde a la influencia de la orografía.

Para la consulta del código íntegro de esta función, el lector puede encontrar en el apéndice C, las secciones C.4., C.5., C.6. y C.7. que contienen dicha función para las 4 orientaciones que permiten obtener las matrices correctoras básicas: viento hacia el Norte, Sur, Este y Oeste. En todas ellas, el código es el mismo exceptuando la carga de la matriz de puntos iniciales en la segunda llamada a Fast Marching, pues su composición depende de la orientación del viento. Sin embargo, se procede a analizar detalladamente aquellas líneas del código que mayor valor poseen para facilitar el entendimiento del lector de la siguiente manera: primero se colocan las líneas de código integras y después los correspondientes comentarios sobre ellas.

```
load sherman_old    % Se carga el mapa

% % SE PREPARAN LOS INPUTS PARA LA PRIMERA LLAMADA A FAST MARCHING

Z2=elevacion(1:300,1:300); % Se elige un trozo, para visualizarlo mejor

Z2=rescale(Z2,1,100); % Las alturas son reescaladas entre 1 y 100
```

- Antes de realizar la expansión de onda de cada punto del suelo llamando a Fast Marching, se tratan los parámetros de entrada para que lleguen de la manera requerida a la función de FM.

- Se carga el mapa deseado y se guarda en la matriz `Z2` la sección de este mapa reescalado con valores entre 0 y 100.

```
W=ones(a,b,h);          % Se iniciliza W con dimensiones 300*300*110. Es
                        % la matriz que representa con 1 y 0 (mapa de
                         % grises) el terreno del mapa en 3D.


IJK=zeros(3,a*b);     % Se inicializa la matriz que almacena
                      % las posiciones iniciales
```

- Se inicializa el mapa de grises (`W`) con unos en todas sus posiciones, por lo que no es un mapa de grises todavía.
- Se inicializa la matriz de puntos iniciales (`IJK`) que tiene 3 filas para guardar las coordenadas de cada punto de inicio.

```
    for i=1:a
        for j=1:b
            for k=1:h
                if k<=round(Z2(i,j))
                    W(i,j,k)=0;
                    IJK(:,a*(i-1)+j)=[i;j;k];
                end
            end
        end
    end
```

Con este bucle recurrente se implementan dos acciones simultáneas:

- Se le asigna 0 a la matriz `W` para crear el mapa de grises en aquellas posiciones que se encuentran en la misma posición que la superficie del terreno del mapa elegido o por debajo de ésta. De tal forma que la matriz `W` posee ceros donde hay tierra (puntos del suelo y por debajo de él) y unos donde hay aire.
- Para la primera expansión (influencia de la orografía en el viento), los puntos iniciales son la superficie terrestre de la sección por lo que se rellena la matriz `IJK` cada vez que en la condición del bucle `if` se evalúa un punto que justamente es una posición del suelo (o capa superficial del terreno).

```
    end_points = Inf;              % Ambos parámetros se establecen
    options.nb_iter_max = Inf;     % como opciones, punto final
                                   % e iteraciones máximas
```

- Se establecen las opciones ofrecidas por el algoritmo, un número infinito de iteraciones y la posición final de la simulación también es infinito.

```
[W3,S] = perform_fast_marching(W, IJK, options);
```

- Se realiza la primera llamada a FM con los parámetros de entrada adecuados y se obtiene el mapa de distancias `W3` que ya cuenta con la expansión de onda desde el suelo, la más relevante de las dos.
- Se propone visualizar la acción llevada a cabo por medio de la figura 4.18.

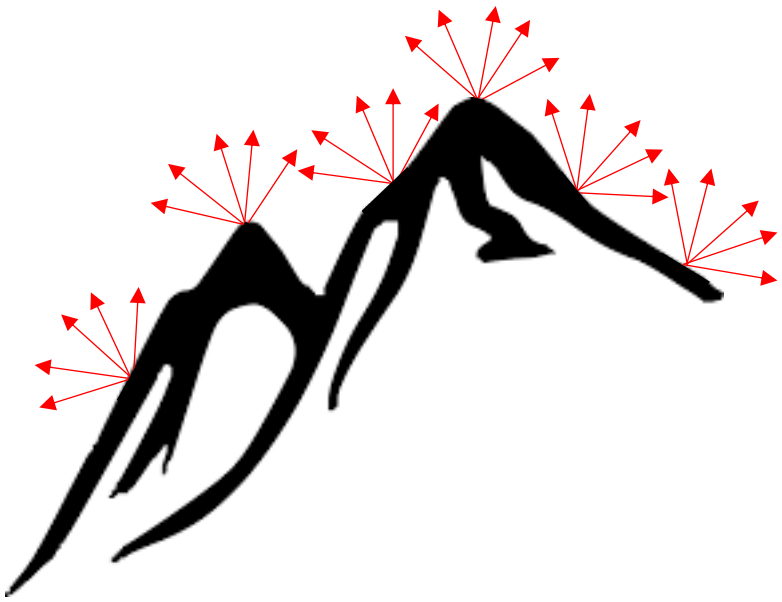


Figura 4.18. Representación gráfica del contenido de `W3` tras la expansión de onda realizada en la primera llamada a FM dentro de la función `prueba_viento`

```
start_points=[];
i=a;
for j=1:b
    for k=1:h
        if k>round(Z2(i,j))
            start_points=[start_points,[i;j;k]];
        end
    end
end
```

Se preparan los *inputs* para la segunda llamada a FM (viento meteorológico), que consiste en lanzar expansiones de onda desde una de las caras laterales de `W3`, el mapa de distancias:

- El primer parámetro de entrada será el mapa de distancias resultante de la primera llamada de FM, `W3`. Esta matriz no será alterada.
- Las opciones de entrada no varían.
- Se rellena la matriz de puntos iniciales (`start_points`) con todos los puntos de una cara lateral. Para cada una de las 4 orientación del viento se utiliza una de las 4 caras laterales. La manera de cargar esta matriz representa la única diferencia en el código de `prueba_viento` para cada orientación. En la figura 4.19. se representa esa diferencia gráficamente.

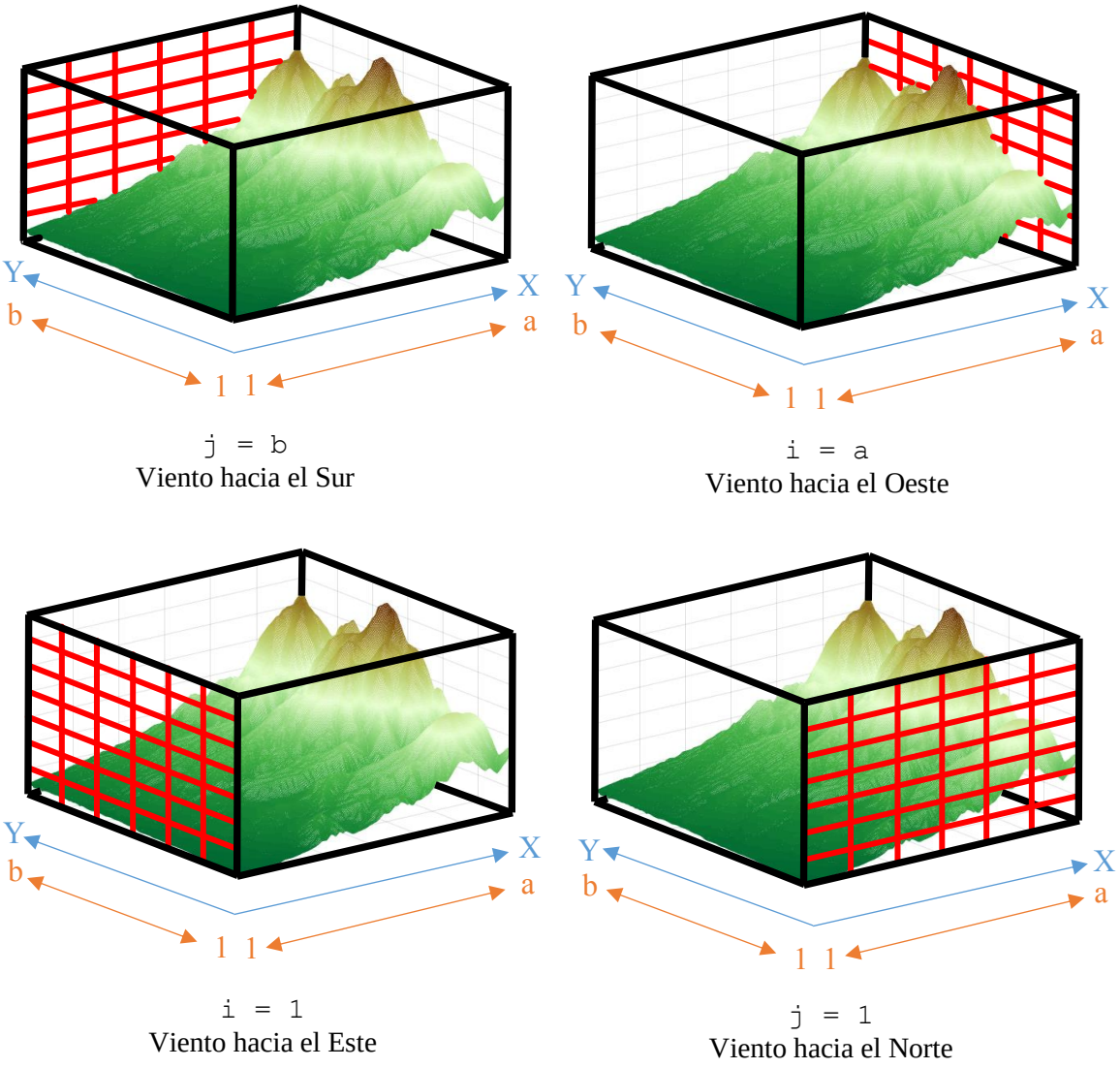


Figura 4.19. Planos de la matriz `start_points` para cada orientación del viento en `prueba_viento`

Sobre la figura 4.19. es importante destacar la línea de código situada debajo de cada dibujo ya que mediante esa asignación a las variables `i` y `j` se establece el plano que va a ser utilizado para formar la matriz de puntos iniciales recorriendo todas sus posiciones. Explicado de otra manera, la cuadrícula roja de cada uno de los 4 dibujos representa los puntos iniciales de la expansión de onda del viento normal por medio de la segunda llamada a Fast Marching.

```
[D,S] = perform_fast_marching(W3, start_points, options);
```

- Se lleva a cabo la segunda llamada a FM y el algoritmo devuelve el mapa de distancias `D` en el que están plasmadas las 2 expansiones de onda de FM.
- Se ofrece una representación gráfica al lector del contenido de `D` en la figura 4.20..

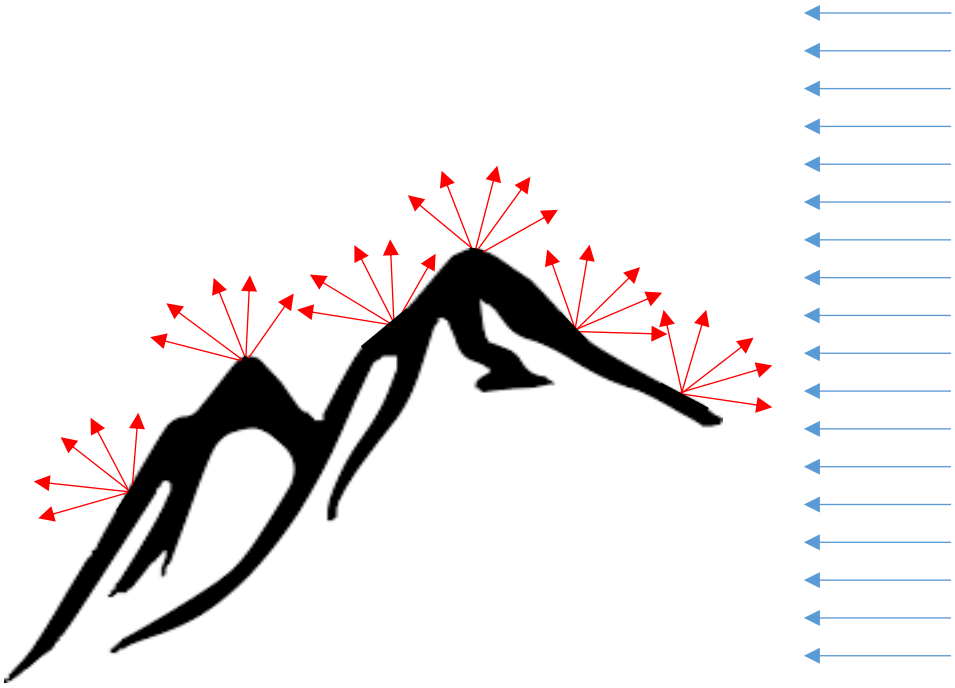


Figura 4.20. Representación gráfica del contenido de `D` tras la expansión de onda realizada en la segunda llamada a FM dentro de la función `prueba_viento`

```
[DX,DY,DZ]=gradient(D);
[X,Y] = meshgrid(1:300,1:300);
Z3=Z2(1:300,1:300);
DX2=zeros(300,300); DY2=zeros(300,300);
for i=1:300;
    for j=1:300;
        DX2(i,j)=DX(i,j,round(Z3(i,j))+4);
        DY2(i,j)=DY(i,j,round(Z3(i,j))+4);
    end
end
```

- Se calcula el gradiente del mapa de distancias `D` obtenido tras la segunda llamada a Fast Marching.
- Mediante los dos bucles `for` recurrentes, se rellenan 2 nuevas matrices (`DX2` y `DY2`) con los valores del gradiente de `D` sumado a la altura del terreno más 4 unidades para visualizar la simulación unos puntos por encima de la superficie del terreno de tal manera que no se incurra en posibles fallos del simulador debido a incongruencias según MatLab.

```
save('Che_EO_X.mat','DX2');
save('Che_EO_Y.mat','DY2');
```

- En última instancia, se guardan las matrices `DX2` y `DY2` debido a que siempre que sean requeridas por sistema para realizar la simulación tendrán el valor inicial con el que son guardadas.

Tras detallar esta función línea por línea, se procede a presentar al lector por medio de las figuras 4.21. y 4.22. su *output*.

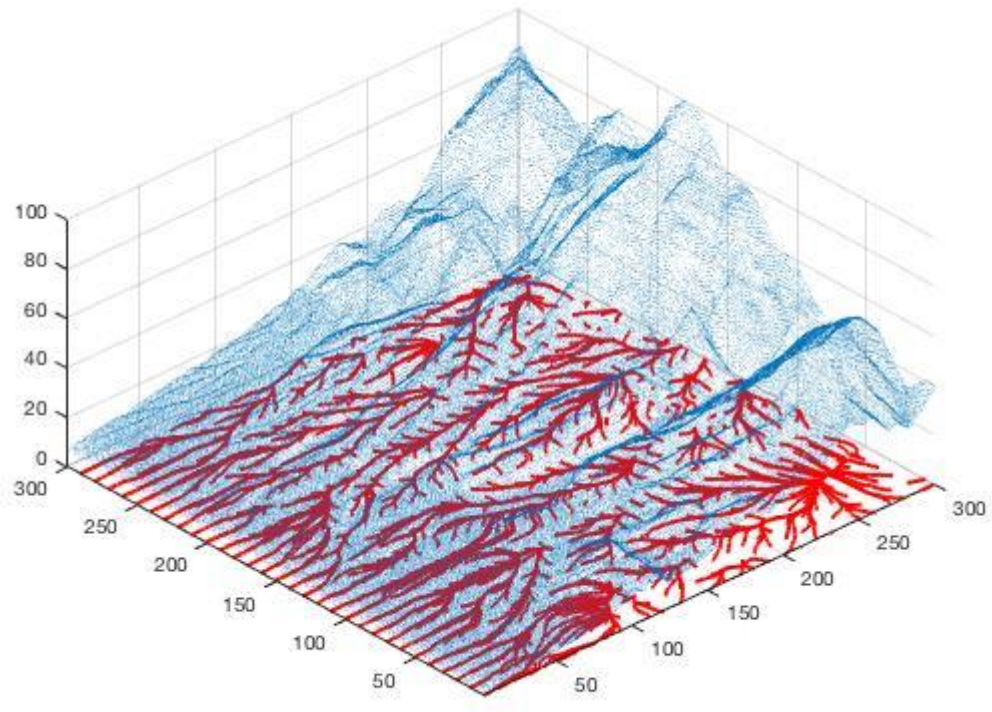


Figura 4.21. Representación 1 del *output* de la función `prueba_viento`

En la figura 4.21. se pueden visualizar los recorridos que siguen las líneas de viento al cruzar el terreno montañoso por medio de las líneas rojas apiladas en el plano inferior (z = 0) resultado de realizar las dos expansiones de onda explicadas en este capítulo por medio de la función `prueba_viento`. Resulta interesante comentar que antes de alcanzar los obstáculos propios del terreno montañoso las líneas de viento rojas parten todas con la misma dirección y paralelas entre sí. Pero, a medida que avanzan, son forzadas a girar hacia los lados para sortear los obstáculos, tal y como ocurre con el viento real.

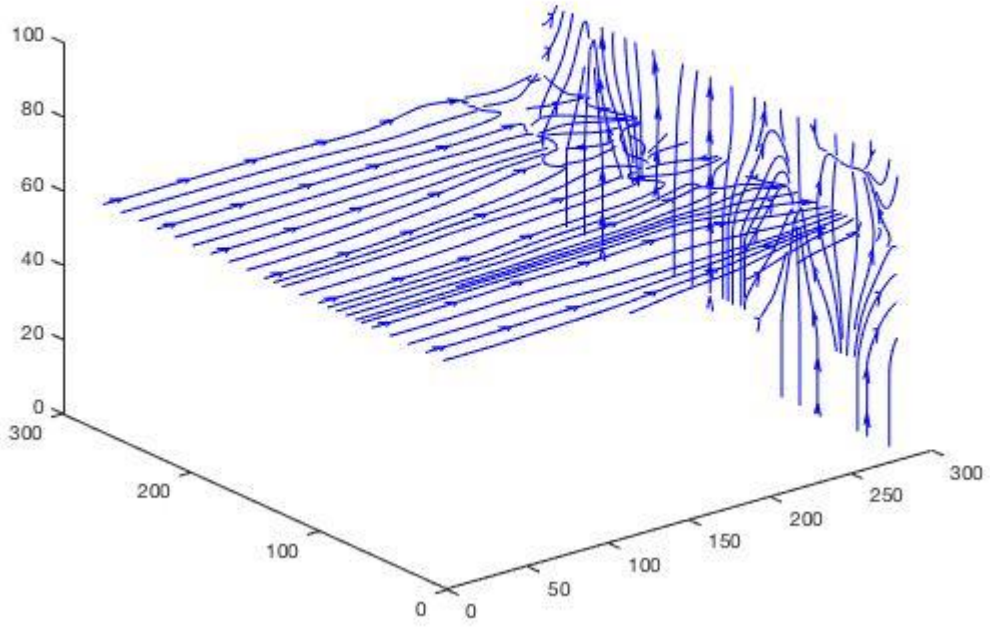


Figura 4.22. Representación 2 del *output* de la función `prueba_viento`

La suerte de utilizar MatLab para desarrollar este simulador permite obtener representaciones muy diversas debido a las amplias posibilidades que ofrece. Es lo que permite que se pueda ofrecer al lector otra representación de la misma información que en la figura 4.21. en la figura 4.22. Como se puede observar, para esta figura se han elegido dos planos aleatorios de la zona montañosa (uno vertical y otro horizontal) con las líneas de viento contenidas en ellas. De tal forma, que se puede observar cómodamente para una altura o profundidad determinadas el comportamiento del viento en relación al terreno que atraviesa. En este caso, se puede concluir que en el plano horizontal las líneas

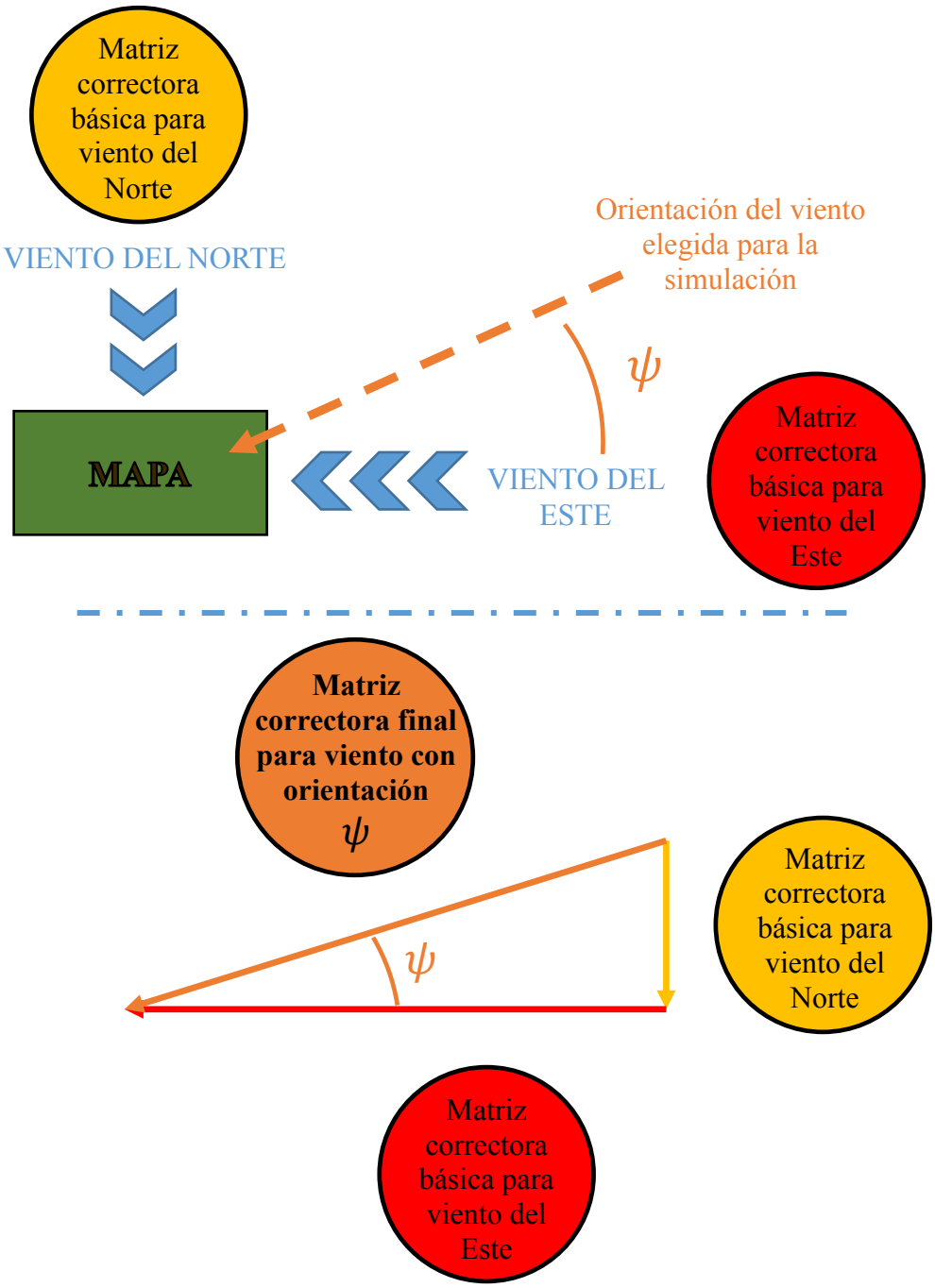
de viento prácticamente se mantienen constantes hasta el final que es donde a esa altura se encuentra la montaña y provoca que se desvíen hacia los lados. Para el plano vertical, se percibe la misma situación, líneas de viento ascendentes que sortean las montañas realizando curvas laterales.

Estas dos representaciones demuestran la calidad de la función `prueba_viento` y confirman su viabilidad para ser utilizadas en la tarea de plasmar la influencia de la orografía en el viento en un escenario de incendio forestal.

4.3.5. Función `vientoORO`

Previamente está función ha sido caracterizada como la coordinadora de la mejora. En ella, el ángulo obtenido con la función `angulo`, las matrices básicas correctoras obtenidas con la función `prueba_viento` y un indicador del mapa seleccionado son parámetros de entrada. Por tanto, sobre esta recae la responsabilidad de crear la matriz correctora final para un ángulo α. Al igual que la función `angulo`, utilizará las propiedades trigonométricas de los triángulos rectángulos, en concreto, la aplicación de senos y cosenos. Dicha matriz correctora final conformará el *output* de `vientoORO`.

El procedimiento trigonométrico de esta función se ilustra con la figura 4.23.



Aplicando las propiedades trigonométricas de los triángulos rectángulos se tiene que:

Figura 4.23. Representación gráfica del método elegido para obtener la matriz correctora para una determinada orientación del viento

Permítase el tratamiento de las matrices correctoras dentro de esta figura como valores métricos debido exclusivamente para realizar una ilustración explicativa de la definición de senos y cosenos dejando claro el conocimiento del alumno de las diferencias entre una longitud y una matriz. Se ha decidido llevar a cabo esta comparación debido a que MatLab durante la simulación realizará la operación matemática de la figura 4.21. donde se suma cada matriz correctora básica que previamente ha sido multiplicada por el seno o coseno del ángulo para formar la matriz correctora final, pero con los valores que hay en cada posición de esas matrices, que, estas sí serán operaciones matemáticas válidas. Sin embargo, para representar este conjunto de operaciones se ha permitido tratar a una matriz como un número.

Para llevar a cabo dicha tarea, se ha construido la función `vientoORO` con dos bucles `if` recurrentes (uno para el mapa Sherman y otro para el Cherokee) de 4 condiciones cada uno. Las 4 condiciones representan los 4 cuadrantes a los que puede pertenecer la orientación del viento. A continuación, se explica el código de esta función con la misma metodología que se ha empleado para analizar la función `prueba_viento`, es decir, primero se establecen las líneas íntegras y después los comentarios correspondientes sobre ellas. Se recuerda al lector que el código completo de la función `vientoORO` se puede consultar en el apéndice C sección C.8.

```
function [vientoOrografia_X, vientoOrografia_Y]=vientoORO(alfa, cheoshe)

load Che_NS_X;    %  Carga la matriz DX1
load Che_NS_Y;    %  Carga la matriz DY1

load She_NS_X;    %  Carga la matriz DX2
load She_NS_Y;    %  Carga la matriz DY2

load Che_OE_X;    %  Carga la matriz DX3
load Che_OE_Y;    %  Carga la matriz DY3

load She_OE_X;    %  Carga la matriz DX4
load She_OE_Y;    %  Carga la matriz DY4

load Che_SN_X;    %  Carga la matriz DX5
load Che_SN_Y;    %  Carga la matriz DY5

load She_SN_X;    %  Carga la matriz DX6
load She_SN_Y;    %  Carga la matriz DY6

load Che_EO_X;    %  Carga la matriz DX7
load Che_EO_Y;    %  Carga la matriz DY7

load She_EO_X;    %  Carga la matriz DX8
load She_EO_Y;    %  Carga la matriz DY8
```

- Como parámetros de entrada, esta función recibe el ángulo y el indicador del mapa.

- El primer paso es cargar las matrices correctoras básicas resultantes de ejecutar `prueba_viento` donde para cada orientación del mapa se forman 2 matrices (una para el eje X y otra para el eje Y), por tanto, como se han utilizado 4 orientaciones del viento (hacia el Norte, Sur, Este y Oeste), para cada mapa se han almacenado previamente 8 matrices correctoras básicas. Dado que por el momento se emplean 2 mapas (Sherman y Cherokee) el resultado final es la carga en la función `vientoORO` de 16 matrices.

- Se cargan 16 archivos .mat que contienen una matriz cada uno. El nombre del archivo es distinto del nombre de la matriz contenida en él, por lo que se presenta al lado de cada línea en el comentario el nombre de la matriz correspondiente.

```
%  BUCLE CHEROKEE

if ((0<=alfa) && (alfa<=90) && (cheoshe==1))
```

- El primer bucle de 4 condiciones es del mapa Cherokee y arriba se presenta la primera condición. Como las 8 condiciones son idénticas estructuralmente se analiza solamente ésta debido a que se considera suficiente para el entendimiento de las 8 condiciones (4 por cada mapa).

- La condición es compuesta. Hay 3 restricciones: las dos primeras limitan la posición del ángulo tanto inferior como superiormente formando el cuadrante I (donde 0º < `alfa` < 90º). La última restricción evalúa el indicador del mapa. Se establecido que cuando la variable `cheoshe` tenga valor 1, el mapa escogido en la simulación por el usuario es Cherokee y cuando valga 0, el mapa Sherman).

```
viEO_X=DX7;
viEO_Y=DX7;

viSN_X=DX5;
viSN_Y=DY5;

vientoOrografia_X = sin(alfa) * viEO_X + cos(alfa) * viSN_X;
vientoOrografia_Y = sin(alfa) * viEO_Y + cos(alfa) * viSN_Y;
```

- Se lleva a cabo la operación trigonométrica ilustrada en la figura 4.21. basada en triángulos rectángulos que tienen la propiedad de que la hipotenusa es igual a la suma del cateto colocado enfrente del ángulo conocido multiplicado por su seno y del cateto adyacente a dicho ángulo multiplicado por su coseno. Se almacena el

valor resultante en las matrices correctoras finales (una para cada eje) `vientoOrografia_X` y `vientoOrografia_Y`.

```
vientoOrografia_X=rescale(vientoOrografia_X,0,1);
vientoOrografia_Y=rescale(vientoOrografia_Y,0,1);
```

- Finalmente, tras obtener las matrices correctoras finales, sus valores son reescalados entre 0 y 1 para que influyan de la manera deseada en el algoritmo, situación que será explicada en el siguiente apartado.

4.3.6. Función ejecuta_FM_prueba_R

El desarrollo de la mejora ya ha sido documentado en los anteriores apartados. Cada elemento clave ha sido dividido en secciones para llegar a cabo explicaciones individuales de código, pero también posicionando cada sección dentro de la estructura, aclarando cuál es su aportación dentro del plan de implementación de la mejora y cómo está conectada con determinadas funciones por medio de los parámetros de entrada, los datos que producen o el matiz que aportan al conjunto. Sin embargo, todo debe quedar plasmado en el código del simulador que previamente existía. Se llevará a cabo en la función `ejecuta_FM_prueba_R` y dentro de este apartado se precisa la localización exacta.

Hasta este punto, se ha dado una visible relevancia a la mejora que se desea llevar a cabo tratando se definirla perfectamente desde distintas perspectivas. Sin embargo, no se ha hecho tanto hincapié en la manera de implementarla. El modo seleccionado es corregir los campos vectoriales del eje X y eje Y con un factor que contabilice el impacto del terreno en la distribución del viento.

En el código del programa antes de comenzar este proyecto de mejora, en concreto en la función `ejecuta_FM_prueba`, uno de los *inputs* que debe recibir Fast Marching para realizar la simulación del incendio sin dar errores son los campos vectoriales `FX` y `FY`, los cuáles se calculan como aparece a continuación:

```
FX = -0.5 * vientoX * ones(dim1) + DX;
minFX = min(min(FX)); maxFX = max(max(FX));
FX = rescale(FX, minFX, maxFX);

FY = 0.5 * vientoY * ones(dim2) + DY;
minFY = min(min(FY)); maxFY = max(max(FY));
FY = rescale(FY, minFY, maxFY);
```

Se puede observar que, en este fragmento de código, `FX` es el resultado de sumar dos matrices (`vientoX` y `DX`). Lo mismo sucede con `FY`. Debido a que `vientoX` es una variable simple de tipo *double* (pudiendo tomar sólo valores enteros comprendidos entre -5 y 5, ambos incluidos) se multiplica por una matriz de unos (`ones(dim1)`) con las dimensiones adecuadas para que se puede realizar la suma de matrices sin errores.

Se exponen a continuación las líneas de código de `ejecuta_FM_prueba_R` que conforman la modificación realizada sobre `ejecuta_FM_prueba` que da lugar a `ejecuta_FM_prueba_R`. Sin embargo, en el apéndice C sección C.9. se puede consultar el código íntegro de la función `ejecuta_FM_prueba_R`.

```
alfa=angulo(vientoX, vientoY);
[viorox, vioroy]=vientoORO(alfa, cheoshe);

FX =-0.5*vientoX*viorox(dim1)+DX;
minFX=min(min(FX));maxFX=max(max(FX));

FX=rescale(FX,minFX,maxFX);

FY=0.5*vientoY*vioroy(dim2)+DY;
minFY=min(min(FY));maxFY=max(max(FY));
FY=rescale(FY,minFY,maxFY);
```

- Primeramente, se llama a la función `angulo` que recibe como parámetros de entrada los valores de los 2 *sliders* de viento de la interfaz gráfica obteniendo el valor del ángulo con el que sopla el viento en la simulación del usuario.

- Después se llama a la función `vientoORO` introduciendo como *inputs* el ángulo y el indicador del mapa seleccionado para obtener las matrices correctoras finales que plasman la influencia de la orografía en el viento tanto para el eje X, `viorox`, como para el eje Y, `vioroy`.

- Por último, se cambia la matriz de unos (`ones(dim1)`) por la matriz correctora de la orografía (`viorox` y `vioroy`) en la fórmula de obtención de `FX` y `FY`, de tal forma que dicho factor corrector supondrá una modificación en la primera matriz del cálculo sobre el valor establecido en el *slider* de viento en la variable `vientoX` directamente proporcional a la situación real de esa posición en el terreno. Por lo que, si esa posición se encuentra en una llanura, el valor inicial de `vientoX` será mínimamente reducido por la matriz correctora. Sin embargo, la posición que se encuentre muy cerca de un obstáculo natural como una montaña, será fuertemente corregida por la matriz `viorox` resultando un valor pequeño en esa posición en relación a la reducción de velocidad del viento cuando se encuentra en su trayecto un obstáculo, que es la casuística que no era tenida en cuenta antes del proyecto de mejora pero que a partir de aquí estará siempre presente.

CAPÍTULO 5

RESULTADOS Y CONCLUSIONES

El proceso de integración de la mejora en el simulador ya ha sido descrito. Es necesario mostrar que el simulador y su mejora funcionan adecuadamente. Se procede, por tanto, durante este capítulo a ofrecer al lector la visualización de distintas simulaciones de incendios donde en cada una se modificará sólo una variable con el objetivo de visibilizar el impacto de cada parámetro en el comportamiento del incendio.

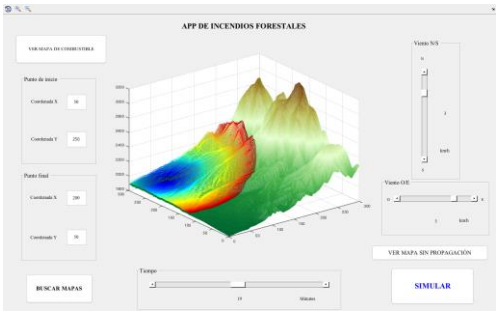
En el primer grupo de simulaciones, varía la posición de inicio y fin de la simulación. El siguiente conjunto visibilizará el impacto del tiempo en el fuego. En última instancia, el comportamiento del viento es la variable que cambia.

Una segunda secuencia de simulaciones estará presente en el capítulo donde se comparan las propagaciones antes de implementar la mejora y después dejando el mismo valor de todos los parámetros. De tal forma, que el lector pueda observar el cambio que produce la mejora y así demostrar la importancia de esta mejora en la búsqueda de la realización de simulaciones lo más reales posible.

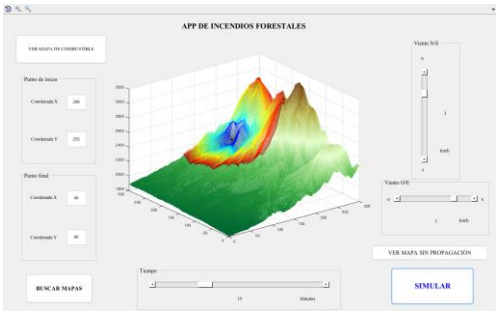
5.1. Simulaciones en el mapa Sherman

- Grupo 1: Se modifica únicamente las posiciones de inicio y fin de la simulación. Los demás parámetros son fijos con los valores que aparecen a continuación:

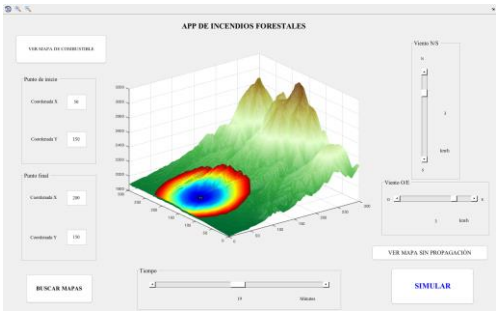
    - Tiempo de propagación: 19 minutos
    - Viento N/S: 3 km/h hacia el Norte
    - Viento E/O: 3 km/h hacia el Este



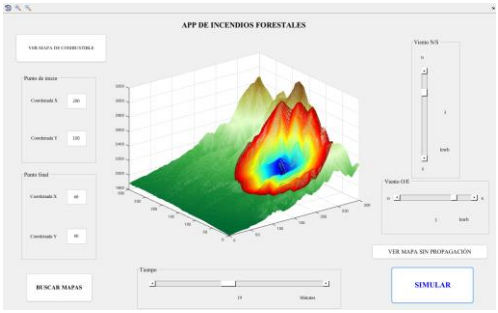
Inicio: (50, 250) Fin: (200, 50)



Inicio: (200, 250) Fin: (60, 80)



Inicio: (50, 150) Fin: (200, 150)

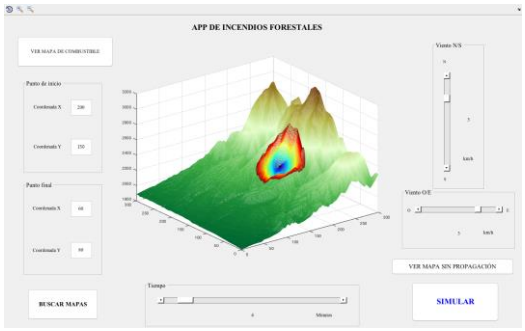


Inicio: (200, 100) Fin: (60, 80)

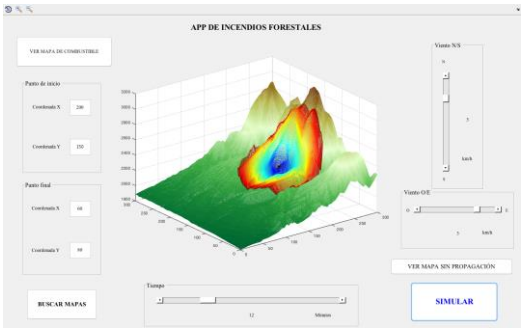
Figura 5.1. Grupo 1 de simulaciones con cambio de los puntos de inicio y fin

- Grupo 2: Se modifica únicamente el tiempo de simulación dejando el resto de los parámetros con los siguientes valores:

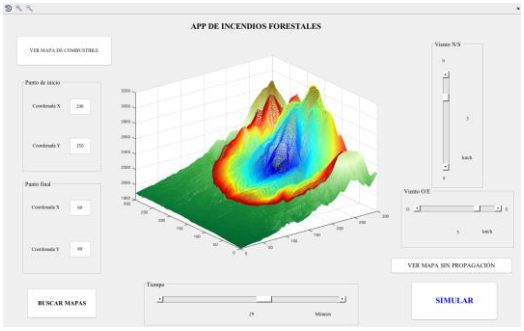
    - Posición de inicio: (200, 150)
    - Posición de final: (60, 80)
    - Viento N/S: 3 km/h hacia el Norte
    - Viento E/O: 3 km/h hacia el Este



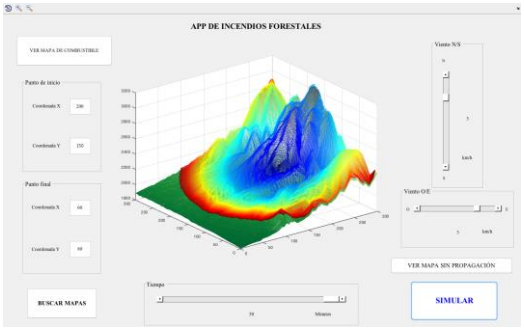
Pasados 4 minutos desde el inicio del incendio



Pasados 12 minutos desde el inicio del incendio



Pasados 29 minutos desde el inicio del incendio



Pasados 50 minutos desde el inicio del incendio

Figura 5.2. Grupo 2 de simulaciones con cambio de tiempo de propagación del incendio

- Grupo 3: Todos los parámetros se mantienen constantes excepto el viento:

    - Tiempo de propagación: 5 minutos
    - Posición de inicio: (150, 170)
    - Posición de final: (50, 50)

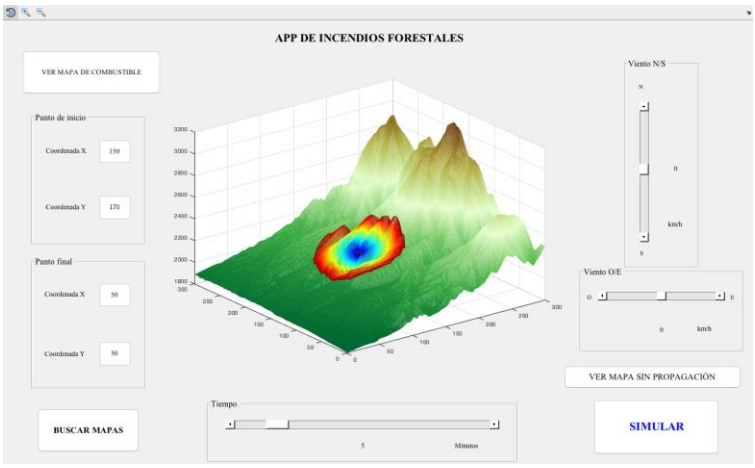
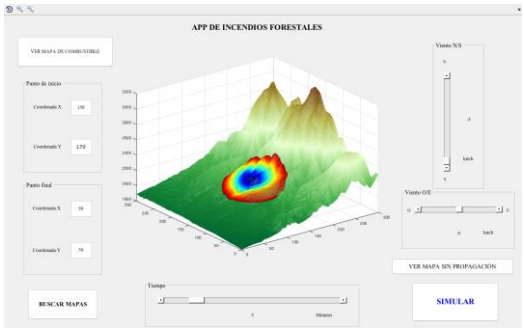
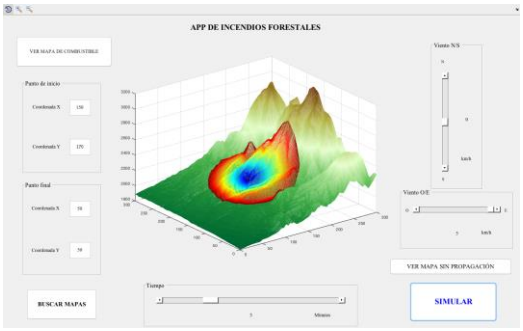


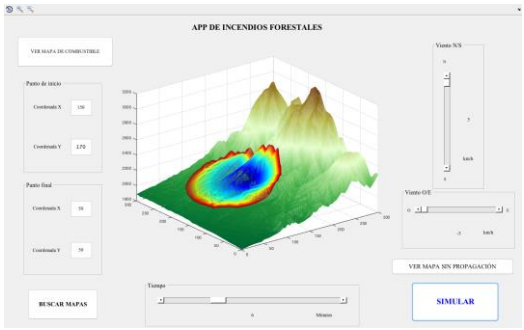
Figura 5.3. Grupo 3 de simulaciones sin tener en cuenta el viento



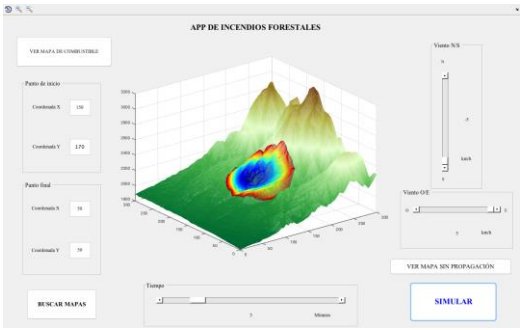
Viento hacia el Sur



Viento hacia el Este



Viento hacia el Noroeste



Viento hacia el Sudeste

Figura 5.4. Grupo 3 de simulaciones con variación del viento

Tras demostrar que el simulador funciona y que todos los parámetros influyen de forma lógica en la propagación simulada, es momento para comparar el programa antes y después de la mejora.

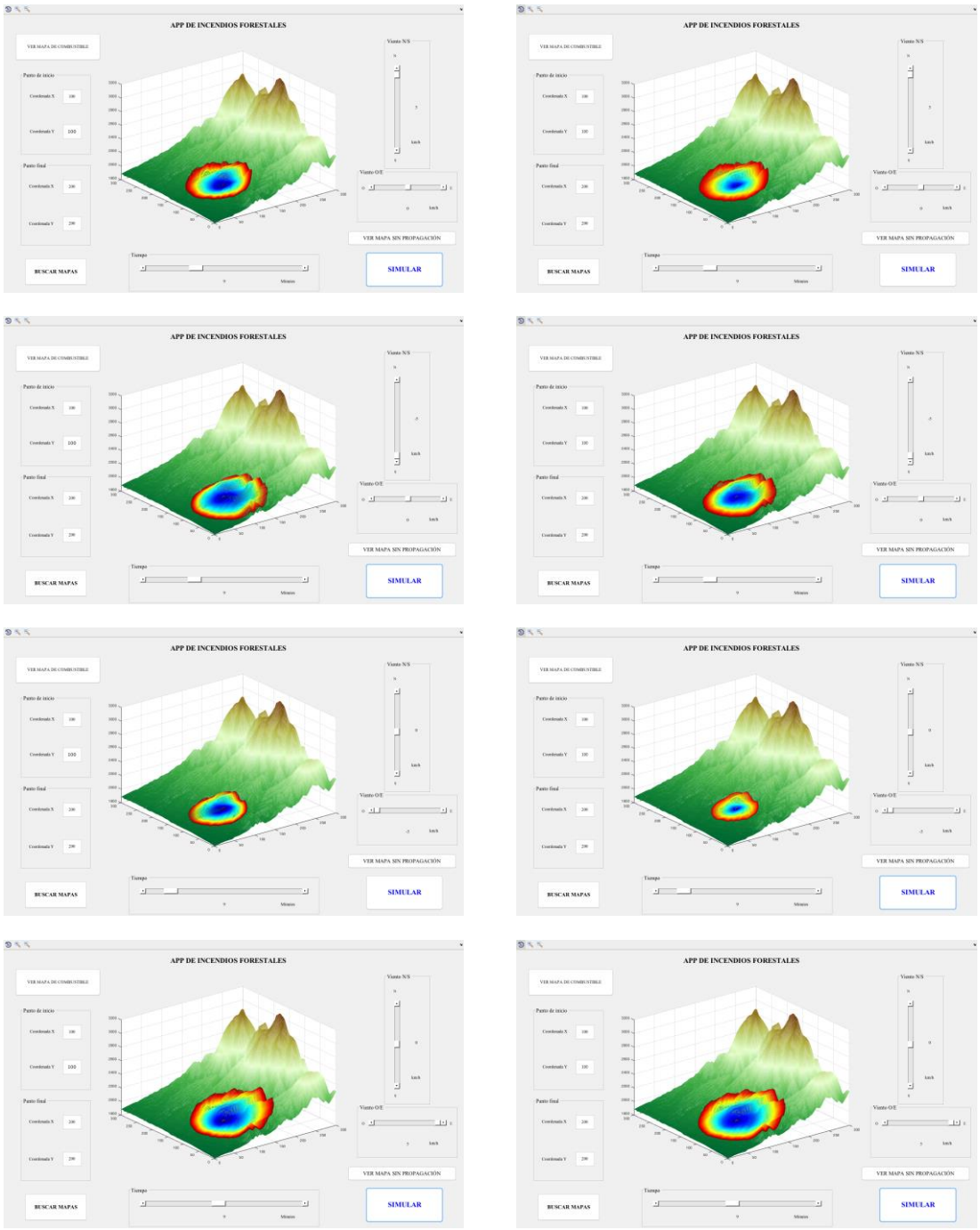


Figura 5.5. Diferencias entre simulaciones antes de la mejora (izquierda) y después (derecha)

Existen algunas orientaciones del viento en las que la simulación es parecida con y sin la mejora, esto es debido a que hay zonas en las que la orografía tiene poco impacto

sobre el viento y prácticamente no lo modifican lo suficiente como para sea fácilmente apreciable.

Sin embargo, cuando la simulación del incendio se realiza en zonas de plena montaña, la mejora se aprecia de una forma más intensa. Como ocurre en las figuras 5.6. y 5.7.

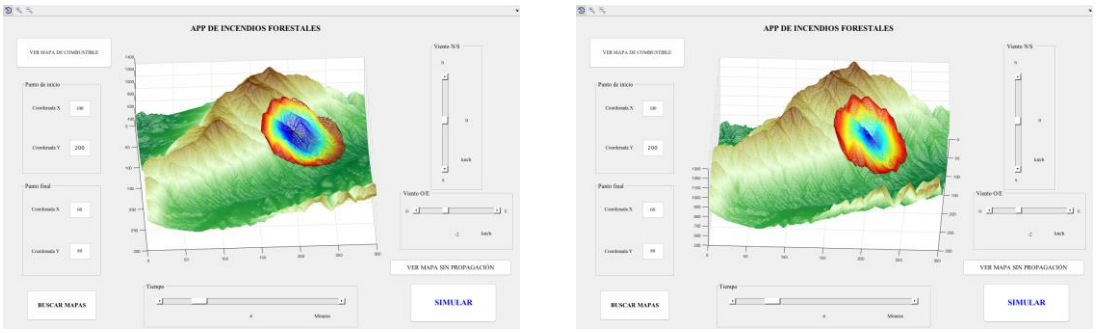


Figura 5.6. Simulaciones antes (izquierda) y después (derecha) de la mejora en el mapa Cherokee I

Figura 5.7. Simulaciones antes (izquierda) y después (derecha) de la mejora en el mapa Cherokee II

5.2. Conclusiones

En primer lugar, corresponde realizar una valoración del estado actual del simulador tras este proyecto de mejora.

Destacan como puntos fuertes la forma en la que se visualiza el resultado final y su sencillo manejo desde el punto de vista del usuario para establecer los parámetros de la simulación. Esto da la posibilidad a cualquier persona, no necesariamente especialista en incendios, de utilizarlo en caso de necesidad para conocer el potencial comportamiento que tendrá un determinado fuego forestal. Por otro lado, debe ser tenido en cuenta el

ahorro computacional y sencillez estructural llevados a cabo durante su diseño, lo que le proporciona al simulador tener una velocidad de cálculo y respuesta diferenciadora en comparación con otros sistemas de previsión de incendios. Cabe recordar que el programa y la mejora están enteramente desarrollados en MatLab sin necesidad de hacer uso de otro software con el que interactuar.

Como elementos de menor potencial de la herramienta, destacan la aproximación de las unidades de medida que se emplean en él y la escasa oferta de funcionalidades de la simulación entre las que deberían ofrecerse el apagado por sí mismo y el avance hacia atrás de un incendio, entre otros, ya que ambos procesos pueden ocurrir en casos de fuegos reales.

Por otra parte, se considera inapropiada la comparación crítica entre esta herramienta y las mejores del mercado pues la inversión de recursos es extremadamente inferior en este proyecto. Sin embargo, hay que remarcar que este proyecto no nace específicamente para desarrollar un simulador potente que compita con los actuales. Se trata de un ensayo para probar el algoritmo Fast Marching como posible motor de un simulador de incendios forestales. Lo que no significa que no puedan detallarse una serie de elementos que el autor considera de mejor calidad que sus homólogos. Entre estas, se encuentra una mayor eficacia computacional que las herramientas Prometheus y Farsite, causado por el empleo del principio de Huygens mientras que el simulador de este proyecto utiliza Fast Marching que es más eficiente. Esto es debido a que el principio de Huygens computa individualmente todas las expansiones de onda del frente de onda y después su envolvente. Otra ventaja es que el usuario ve elementos más sencillos e intuitivos que en la herramienta Farsite (explicada en el capítulo 2) así como se simplifican claramente los pasos que debe seguir el usuario para llevar a cabo su simulación.

En lo que respecta a la mejora, ha quedado demostrado su impacto en los resultados del programa. Pese a que en algunos casos es poco notable, todo el proceso de desarrollo respalda la aportación en el método resolutivo del programa y el salto de calidad experimentado de cara a acercarse al objetivo final del simulador que es conseguir que sea lo más real posible. En concreto, el objetivo de este Trabajo de Fin de grado era incluir la influencia de la orografía en el comportamiento del viento durante un incendio forestal y en vista a los resultados obtenidos, queda comprobado que se ha conseguido satisfactoriamente dicho propósito. Por tanto, se puede continuar con los próximos proyectos de mejora o perfeccionamiento sobre este simulador en otras áreas. Paso a paso, con la aportación del departamento de Sistemas y Automática de la Universidad Carlos III de Madrid junto a los estudiantes seleccionados, irá creciendo este ambicioso proyecto conjunto donde ha quedado confirmada la viabilidad del uso de Fast Marching en el sector de los incendios forestales y del que se espera un producto final listo para ser lanzado al mercado de este tipo de herramientas contribuyendo a la erradicación de las pérdidas tanto humanas como materiales que dejan cada año en nuestro país estas catástrofes naturales, los incendios forestales.

CAPÍTULO 6

FUTURAS LÍNEAS DE MEJORA

De cara a establecer una ruta para las próximas personas que continúen con la evolución del simulador, se enumeran las posibles líneas de mejora a seguir para lograr la herramienta definitiva:

- Las unidades: Por el momento no es proporcional la relación entre las unidades de los parámetros que el usuario establece en la interfaz gráfica con el tamaño de las áreas que, según la simulación, han sido quemadas. El algoritmo en el que se basa el simulador, Fast Marching, es matemático, es decir, utiliza valores sin unidades. Los algoritmos experimentales si son capaces de mantener estas correlaciones de unidad. Por tanto, partiendo de los experimentos, la línea de investigación a seguir para implementar esta mejora debería estar relacionada con el método de estudio de incendios reales en los que se conocen específicamente todos los parámetros que influyeron en él, velocidad del viento, su orientación, puntos inicial y final, tiempo de propagación. Con vistas a recrear ese mismo incendio en el simulador y establecer manualmente factores correctores en el algoritmo que, al ejecutar la simulación, se observe por pantalla, la misma propagación del incendio que ocurrió en la realidad. Es realizar un proceso invertido de cálculo numérico realizado también en otras áreas científicas por medio de la integración de factores correctores en las fórmulas matemáticas. Concretamente, habría que comparar las líneas de viento que produce el programa con datos reales de vientos en montañas para establecer rangos de valores más fiables para cada parámetro.

- Fast Marching: Existe margen de mejora en las líneas de código del algoritmo en torno al cual se construye esta herramienta. Además, las limitaciones de Fast Marching provocan que algunos comportamientos propios de incendios no puedan ser replicados por este simulador como el autoapagado o el avance hacia atrás, ambas ya comentadas. La línea que seguir en la mejora de este elemento se fundamenta en escribir líneas de código para integrar estas funcionalidades en el algoritmo Fast Marching, más concretamente en la función adaptada para MatLab `fast_marching_vectorial_2D` o bien en el caso del retroceso del incendio (que FM no lo puede plasmar de ninguna manera) se podría buscar otro algoritmo o función que proporcionará conjuntos de datos admitidos por FM y que de alguna manera implementaran ese retroceso del incendio. Es el caso del algoritmo Level Set, el cuál permite el uso de valores negativos para poder simular el retroceso o autoapagado del incendio. Un ejemplo de programa que utiliza el algoritmo Level Set es Comsol especializado en resolver ecuaciones diferenciales parciales.

- Apoyo al usuario: Una serie de mejoras podrían implementarse de cara a dar apoyo al usuario en el momento de interacción con el simulador. Entre ellas, se propone dar la posibilidad de cargar el mapa que se quiera sin preocuparse del tipo de archivo (.mat) que es admitido por el programa. Otra podría ser ofrecer una ventana emergente de ayuda, con el que marcar la ruta si fuese necesario y ofrecer comentarios propicios al usuario durante la interacción. Por último, sería aconsejable crear mensajes de error personalizados a cada acción que MatLab no permite para el aumento de información del usuario. Sirva como ejemplo, cuando se establecen coordenadas que sobrepasan el límite de la sección de mapa, el usuario reciba un mensaje con tal problemática y pueda actuar en consecuencia más rápidamente.

- Interfaz: MatLab es una herramienta potente en cálculos matriciales. Sin embargo, su capacidad gráfica de diseño no está igualmente desarrollada. En consecuencia, la interfaz gráfica de usuario es poco estructurada o profesional y mucho menos atractiva artísticamente. Esta línea de mejora futura no es tan relevante como otras en la lista, pero sí que sería interesante, en caso de llegar al simulador definitivo con todas las otras mejoras integradas, realizar una migración a un entorno con mayores opciones de diseño para crear un producto que, visualmente sea apropiado.

- Variables: Pese a que la aplicación cuenta con varios parámetros integrados, se podrían incluir más de carácter meteorológico, temporal u horario. Introducir como *input* posibles lluvias en medio del incendio u olas de sequía, visualizar círculos concéntricos de la simulación estableciendo en cada círculo la hora en la que el frente tiene ese tamaño o establecer fechas horarias reales para poder diferenciar entre un incendio que es iniciado por la mañana y otro que aparece por la noche.

- Código: En su carácter más técnico, se podría someter el código entero a una prueba de economía computacional y eficiencia por parte de un programador experto debido a que el código ha sido desarrollado por estudiantes con conocimientos básicos de programación. Este tratado del código permitiría una velocidad mayor de ejecución a pesar de que actualmente cada simulación se lleva a cabo en apenas unos segundos.

APÉNDICES

Apéndice A

Marco Regulador

No se puede aplicar un marco regulador a este trabajo, ni económico ni legal, ya que se basa en una mejora técnica concreta sobre la propuesta de un programa informático para la simulación de incendios forestales. Incluso, si la base del proyecto fuera la propia creación del simulador tampoco estaría sujeto a ningún marco de regulación ya que se este se emplearía sobre las acciones propias del apagado de un incendio, pero nunca sobre un software de predicción.

Apéndice B

Entorno socio-Económico

Este proyecto de mejora que es estrictamente la integración de un factor corrector en una prueba de viabilidad de un simulador empleando un algoritmo desconocido en el sector no genera ningún tipo de coste y beneficio dado que en ningún momento se ha comercializado ni publicitado, el estado actual del simulador es proyecto universitario en desarrollo que tiene un amplio margen de mejora en varias vertientes diferentes.

Suponiendo el caso de que se ejecutara la lista de mejoras propuestas en el capítulo 6 y posteriormente se comercializara, los beneficios de utilizar el simulador en incendios reales incurriría en un conjunto de beneficios que englobaría recursos materiales que se han dejado de utilizar por culpa de la eficacia del simulador, los recursos económicos no invertidos asociados a los materiales y recursos naturales (indirectamente son recursos económicos) y humanos que no han sido arrasados por las llamas y pueden seguir siendo valiosos y cumplir sus funciones. Este impacto está referido al simulador completo, no específicamente al impacto de la mejora de este trabajo, pero ésta forma parte del simulador en su máximo desarrollo. Es por esto por lo que se considera oportuno plasmar dicho impacto en este documento, así como ofrecer al lector una visualización del presupuesto del simulador por medio de la tabla B.1.

Descripción	**Cantidad**	**Precio Unitario**	**Coste**
Licencia académica individual de MatLab	1 unidad	$500\ ^{€}/_{unidad}$	500 €
Horas trabajadas	300 horas	$15\ ^{€}/_{hora}$	4500 €
Total neto			5000 €
IVA (21%)			1050 €
Total			**6050 €**

Tabla B.1: Presupuesto del simulador

Apéndice C

Códigos

C.1. Algoritmo de Dijkstra

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% EJEMPLO SENCILLO DE LA IMPLEMENTACION DEL ALGORITMO DE DIJKSTRA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Se tiene una malla de 3x3 para que sea más simple

n=3;
% Contiene los vectores de desplazamiento en cualquiera de las 4
% direcciones de la malla

neigh = [[1;0] [-1;0] [0;1] [0;-1];

% Establece la fila en la que se encuentra el punto que se ejija

boundary = @(x)mod(x-1,n) +1;


% Las siguientes funciones hacen posible que dado un punto de la malla k,
% se mueva en dirección i siendo i cada uno de los vectoresd de dirección

ind2sub1 = @(k)[rem(k-1,n)+1;(k - rem(k-1,n) -1)/n + 1];
sub2ind1 = @(u)(u(2)-1)*n+u(1);
Neigh = @(k,i) sub2ind1 (boundary(ind2sub1(k)+neight(:,i)));

% Se obliga a la matriz de costes a tener valor 1 en todos sus nodos
W = ones(n);

% Se establece el punto medio de la malla como posición inicial

x0 = [n/2;n/2];

% Se inicializa D, con todos los puntos desconocidos (distancia infinito)
% excepto el punto inicial

I = sub2ind1(x0);
D = zeros(n)+Inf;
D(I) = 0;

% Igualmente se inicializa la matriz que informa sobre el estado de cada
% posición. Todos 0 excepto el de inicio que vale 1.

S = zeros(n);
S(I) = 1;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Comienzo de la parte iterativa
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Se actualizan que pasen al estado "muerto" aquellos puntos que ya han
% alcanzados por la expansión de la onda
[tmp,j] = sort(D(I)); j = j(1);
i = I(j); I(j)=[];
S(i) = -1;

% Ahora se consideran los puntos "vecinos" descartando los "muertos"

J = [Neigh(i,1); Neigh(i,2); Neigh(i,3); Neigh(i,4)];
J(S(J)==-1) = [];

% El siguiente paso es añadir en la matriz S aquellos puntos con estado
% "vecino" que no hayan sido considerados todavía

J1=J(S(J)==0);
I = [I;J1];
S(J1) = 1;

% Por último, se realiza la actualización de los valores del mapa de
% distancias D

for j=J'
    dx = min( D([Neigh(j,1) Neigh(j,2)]) );
    dy = min( D([Neigh(j,3) Neigh(j,4)]) );
    D(j) = min (dx+W(j), dy+W(j));
end
```

C.2. Algoritmo de Fast Marching

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CASO SIMPLE DE LA IMPLEMENTACIÓN DE FAST MARCHING
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Seleccionamos matriz 3x3 por simplicidad

n=3;

% Se almacenan los vectores de desplazamiento en las 4 direcciones

neigh = [[1;0] [-1;0] [0;1] [0;-1];

% Establece la fila en la que se encuentra el punto que se ejija
```

```
boundary = @(x)mod(x-1,n) +1;


% Las siguientes funciones hacen posible que dado un punto de la malla k,
% se mueva en dirección i siendo i cada uno de los vectores de dirección

ind2sub1 = @(k)[rem(k-1,n)+1;(k - rem(k-1,n) -1)/n + 1];
sub2ind1 = @(u)(u(2)-1)*n+u(1);
Neigh = @(k,i) sub2ind1 (boundary(ind2sub1(k)+neight(:,i)));

% Se obliga a la matriz de costes a tener valor 1 en todos sus nodos
W = ones(n);

% Se establece el punto medio de la malla como posición inicial

x0 = [n/2;n/2];

% Se inicializa D, con todos los puntos desconocidos (distancia infinito)
% excepto el punto inicial

I = sub2ind1(x0);
D = zeros(n)+Inf;
D(I) = 0;

% Igualmente se inicializa la matriz que informa sobre el estado de cada
% posición. Todos 0 excepto el de inicio que vale 1.

S = zeros(n);
S(I) = 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Comienzo de la parte iterativa
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Se actualizan que pasen al estado "muerto" aquellos puntos que ya han
% alcanzados por la expansión de la onda
[tmp,j] = sort(D(I)); j = j(1);
i = I(j); I(j)=[];
S(i) = -1;

% Ahora se consideran los puntos "vecinos" descartando los "muertos"

J = [Neigh(i,1); Neigh(i,2); Neigh(i,3); Neigh(i,4)];
J(S(J)==-1) = [];

% El siguiente paso es añadir en la matriz S aquellos puntos con estado
% "vecino" que no hayan sido considerados todavía

J1=J(S(J)==0);
I = [I;J1];
S(J1) = 1;

% Por último, se realiza la actualización de los valores del mapa de
% distancias D
```

```
for j=J'
    dx = min( D([Neigh(j,1) Neigh(j,2)]) );
    dy = min( D([Neigh(j,3) Neigh(j,4)]) );
    Delta = 2*W(j) - (dx-dy)^2;
    if abs(dx-dy)<=W(j)
        D(j) = (dx+dy+sqrt(Delta))/2;
    else
        D(j) = min(dx+W(j), dy+W(j));
    end
end
```

C.3. Función angulo

```
function alfa=angulo(vientoX, vientoY)


%Cuadrante I
if((vientoX<0) && (vientoY>0))

    alfa=atand(-vientoX / vientoY);

%Cuadrante II
elseif((vientoX <0) &&( vientoY <0))

    alfa=180-atand(-vientoX /- vientoY);

%Cuadrante III
elseif ((vientoX >0) && (vientoY <0))

    alfa=180+atand(vientoX /- vientoY);

%Cuadrante IV
elseif ((vientoX >0) && (vientoY >0))

    alfa=360-atand(vientoX / vientoY);

%Viento hacia el Norte
elseif((vientoX ==0) && (vientoY >0))

    alfa=0;

%Viento hacia el Sur
elseif((vientoX ==0) && (vientoY <0))

    alfa=180;

%Viento hacia el Oeste
```

```
elseif((vientoX <0) && (vientoY ==0))

    alfa=90;

%Viento hacia el Este
elseif((vientoX >0) && (vientoY ==0))

    alfa=270;

end


alfa=round(alfa);


end
```

C.4. Función `prueba_viento` para viento Este - Oeste

```
    load sherman_old    % Se carga el mapa

    % % SE PREPARAN LOS INPUTS PARA LA PRIMERA LLAMADA A FAST MARCHING

    Z2=elevacion(1:300,1:300); % Se elige un trozo, para visualizarlo mejor

    Z2=rescale(Z2,1,100); % Las alturas son reescaladas entre 1 y 100

    [a,b]=size(Z2);

    h=110;
    W=ones(a,b,h);     % Se iniciliza W con dimensiones 300*300*110. Es
                       % la matriz que representa con 1 y 0 (mapa de grises)
                       % el terreno del mapa en 3D.

    IJK=zeros(3,a*b);     % Se inicializa la matriz que almacena
                          % las posiciones iniciales


    % Se procede a rellenar la matriz W en la que toda la superficie del
    % terreno junto con las posiciones de debajo tendran valor 0. Aquellas
    % posiciones que sean aire en el mapa tendrán valor 1.

    for i=1:a
        for j=1:b
            for k=1:h
                if k<=round(Z2(i,j))
                    W(i,j,k)=0;
```

```
                IJK(:,a*(i-1)+j)=[i;j;k];
            end
        end
    end
end


end_points = Inf;              % Ambos parámetros se establecen
options.nb_iter_max = Inf;     % como opciones, punto final
                               % e iteraciones maximas


% Se procede a llamar la primera vez a la funcion FM para simular la
% expansion de una onda en cada punto de la superficie del terreno:

[W3,S] = perform_fast_marching(W, IJK, options);


W3(isinf(W3))=0;
W3(isnan(W3))=0;
W3=smooth3(W3);
W3=rescale(W3,0,1);  % Aqui hay que cambiar el rescalado segun la
                     % fuerza que deseemos. Al final corresponde a
                     % la corriente ascendente debida al incendio.

W3=min(W,W3);

 % % SE PREPARAN LOS INPUTS PARA LA SEGUNDA LLAMADA A FAST MARCHING



start_points=[];
i=a;
for j=1:b
    for k=1:h
        if k>round(Z2(i,j))
            start_points=[start_points,[i;j;k]];
        end
    end
end


% Se llama a Fast Marching para producir expansiones de onda en cada
% punto de la cara Este de W3 sobre la matriz W3 que ya cuenta con la
% expansión de onda de cada punto de la superficie. De tal forma que el
% output de esta segunda llamada será una matriz de distancias que
% cuenta con 2 expansiones de onda.

[D,S] = perform_fast_marching(W3, start_points, options);
D(isinf(D))=0;
D(isnan(D))=0;
D=smooth3(D);
```

```
[DX,DY,DZ]=gradient(D);
DX(isnan(DX))=0;DY(isnan(DY))=0;DZ(isnan(DZ))=0;
[X,Y] = meshgrid(1:300,1:300);
Z3=Z2(1:300,1:300);
DX2=zeros(300,300); DY2=zeros(300,300); DZ2=zeros(300,300);

for i=1:300;
    for j=1:300;
        DX2(i,j)=DX(i,j,round(Z3(i,j))+4);
        DY2(i,j)=DY(i,j,round(Z3(i,j))+4);
        DZ2(i,j)=DZ(i,j,round(Z3(i,j))+4);
    end
end

% En esta línea ya se han rellenado las matrices que contienen la
% informacion sobre las distancias tanto para el eje X (DX2) como para
% el eje Y (DY2) y se procede a guardarlas para poder hacer uso de
% ellas en cada simulación directamente en lugar de tener que correr
% esta función y rellenarlas cada vez:

save('Che_EO_X.mat','DX8');
save('Che_EO_Y.mat','DY8');
```

C.5. Función `prueba_viento` para viento Oeste - Este

```
load sherman_old    % Se carga el mapa

% % SE PREPARAN LOS INPUTS PARA LA PRIMERA LLAMADA A FAST MARCHING

Z2=elevacion(1:300,1:300); % Se elige un trozo, para visualizarlo mejor

Z2=rescale(Z2,1,100); % Las alturas son reescaladas entre 1 y 100

[a,b]=size(Z2);

h=110;
W=ones(a,b,h);     % Se iniciliza W con dimensiones 300*300*110. Es
                   % la matriz que representa con 1 y 0 (mapa de grises)
                   % el terreno del mapa en 3D.

IJK=zeros(3,a*b);     % Se inicializa la matriz que almacena
                      % las posiciones iniciales


% Se procede a rellenar la matriz W en la que toda la superficie del
% terreno junto con las posiciones de debajo tendran valor 0. Aquellas
% posiciones que sean aire en el mapa tendrán valor 1.

for i=1:a
```

```
    for j=1:b
        for k=1:h
            if k<=round(Z2(i,j))
                W(i,j,k)=0;
                IJK(:,a*(i-1)+j)=[i;j;k];
            end
        end
    end
end


end_points = Inf;              % Ambos parámetros se establecen
options.nb_iter_max = Inf;     % como opciones, punto final
                               % e iteraciones maximas


% Se procede a llamar la primera vez a la funcion FM para simular la
% expansion de una onda en cada punto de la superficie del terreno:

[W3,S] = perform_fast_marching(W, IJK, options);


W3(isinf(W3))=0;
W3(isnan(W3))=0;
W3=smooth3(W3);
W3=rescale(W3,0,1);  % Aqui hay que cambiar el rescalado segun la
                     % fuerza que deseemos. Al final corresponde a
                     % la corriente ascendente debida al incendio.

W3=min(W,W3);

 % % SE PREPARAN LOS INPUTS PARA LA SEGUNDA LLAMADA A FAST MARCHING



start_points=[];
i=1;
for j=1:b
    for k=1:h
        if k>round(Z2(i,j))
            start_points=[start_points,[i;j;k]];
        end
    end
end


% Se llama a Fast Marching para producir expansiones de onda en cada
% punto de la cara Este de W3 sobre la matriz W3 que ya cuenta con la
% expansión de onda de cada punto de la superficie. De tal forma que el
% output de esta segunda llamada será una matriz de distancias que
% cuenta con 2 expansiones de onda.

[D,S] = perform_fast_marching(W3, start_points, options);
D(isinf(D))=0;
```

```
D(isnan(D))=0;
D=smooth3(D);


[DX,DY,DZ]=gradient(D);
DX(isnan(DX))=0;DY(isnan(DY))=0;DZ(isnan(DZ))=0;
[X,Y] = meshgrid(1:300,1:300);
Z3=Z2(1:300,1:300);
DX2=zeros(300,300); DY2=zeros(300,300); DZ2=zeros(300,300);

for i=1:300;
    for j=1:300;
        DX2(i,j)=DX(i,j,round(Z3(i,j))+4);
        DY2(i,j)=DY(i,j,round(Z3(i,j))+4);
        DZ2(i,j)=DZ(i,j,round(Z3(i,j))+4);
    end
end

% En esta línea ya se han rellenado las matrices que contienen la
% informacion sobre las distancias tanto para el eje X (DX2) como para
% el eje Y (DY2) y se procede a guardarlas para poder hacer uso de
% ellas en cada simulación directamente en lugar de tener que correr
% esta función y rellenarlas cada vez:

save('Che_EO_X.mat','DX4');
save('Che_EO_Y.mat','DY4');
```

C.6. Función `prueba_viento` para viento Norte - Sur

```
load sherman_old    % Se carga el mapa

% % SE PREPARAN LOS INPUTS PARA LA PRIMERA LLAMADA A FAST MARCHING

Z2=elevacion(1:300,1:300); % Se elige un trozo, para visualizarlo mejor

Z2=rescale(Z2,1,100); % Las alturas son reescaladas entre 1 y 100

[a,b]=size(Z2);

h=110;
W=ones(a,b,h);     % Se iniciliza W con dimensiones 300*300*110. Es
                   % la matriz que representa con 1 y 0 (mapa de grises)
                   % el terreno del mapa en 3D.

IJK=zeros(3,a*b);     % Se inicializa la matriz que almacena
                      % las posiciones iniciales


% Se procede a rellenar la matriz W en la que toda la superficie del
```

```
% terreno junto con las posiciones de debajo tendran valor 0. Aquellas
% posiciones que sean aire en el mapa tendrán valor 1.

for i=1:a
    for j=1:b
        for k=1:h
            if k<=round(Z2(i,j))
                W(i,j,k)=0;
                IJK(:,a*(i-1)+j)=[i;j;k];
            end
        end
    end
end


end_points = Inf;              % Ambos parámetros se establecen
options.nb_iter_max = Inf;     % como opciones, punto final
                               % e iteraciones maximas


% Se procede a llamar la primera vez a la funcion FM para simular la
% expansion de una onda en cada punto de la superficie del terreno:

[W3,S] = perform_fast_marching(W, IJK, options);


W3(isinf(W3))=0;
W3(isnan(W3))=0;
W3=smooth3(W3);
W3=rescale(W3,0,1);  % Aqui hay que cambiar el rescalado segun la
                     % fuerza que deseemos. Al final corresponde a
                     % la corriente ascendente debida al incendio.

W3=min(W,W3);

 % % SE PREPARAN LOS INPUTS PARA LA SEGUNDA LLAMADA A FAST MARCHING



start_points=[];
j=b;
for i=1:a
    for k=1:h
        if k>round(Z2(i,j))
            start_points=[start_points,[i;j;k]];
        end
    end
end


% Se llama a Fast Marching para producir expansiones de onda en cada
% punto de la cara Este de W3 sobre la matriz W3 que ya cuenta con la
% expansión de onda de cada punto de la superficie. De tal forma que el
% output de esta segunda llamada será una matriz de distancias que
```

```
% cuenta con 2 expansiones de onda.

[D,S] = perform_fast_marching(W3, start_points, options);
D(isinf(D))=0;
D(isnan(D))=0;
D=smooth3(D);


[DX,DY,DZ]=gradient(D);
DX(isnan(DX))=0;DY(isnan(DY))=0;DZ(isnan(DZ))=0;
[X,Y] = meshgrid(1:300,1:300);
Z3=Z2(1:300,1:300);
DX2=zeros(300,300); DY2=zeros(300,300); DZ2=zeros(300,300);

for i=1:300;
    for j=1:300;
        DX2(i,j)=DX(i,j,round(Z3(i,j))+4);
        DY2(i,j)=DY(i,j,round(Z3(i,j))+4);
        DZ2(i,j)=DZ(i,j,round(Z3(i,j))+4);
    end
end

% En esta línea ya se han rellenado las matrices que contienen la
% informacion sobre las distancias tanto para el eje X (DX2) como para
% el eje Y (DY2) y se procede a guardarlas para poder hacer uso de
% ellas en cada simulación directamente en lugar de tener que correr
% esta función y rellenarlas cada vez:

save('Che_EO_X.mat','DX2');
save('Che_EO_Y.mat','DY2');
```

C.7. Función `prueba_viento` para viento Sur - Norte

```
load sherman_old    % Se carga el mapa

% % SE PREPARAN LOS INPUTS PARA LA PRIMERA LLAMADA A FAST MARCHING

Z2=elevacion(1:300,1:300); % Se elige un trozo, para visualizarlo mejor

Z2=rescale(Z2,1,100); % Las alturas son reescaladas entre 1 y 100

[a,b]=size(Z2);

h=110;
W=ones(a,b,h);     % Se iniciliza W con dimensiones 300*300*110. Es
                   % la matriz que representa con 1 y 0 (mapa de grises)
                   % el terreno del mapa en 3D.
```

```
IJK=zeros(3,a*b);    % Se inicializa la matriz que almacena
                     % las posiciones iniciales


% Se procede a rellenar la matriz W en la que toda la superficie del
% terreno junto con las posiciones de debajo tendran valor 0. Aquellas
% posiciones que sean aire en el mapa tendrán valor 1.

for i=1:a
    for j=1:b
        for k=1:h
            if k<=round(Z2(i,j))
                W(i,j,k)=0;
                IJK(:,a*(i-1)+j)=[i;j;k];
            end
        end
    end
end


end_points = Inf;              % Ambos parámetros se establecen
options.nb_iter_max = Inf;     % como opciones, punto final
                               % e iteraciones maximas


% Se procede a llamar la primera vez a la funcion FM para simular la
% expansion de una onda en cada punto de la superficie del terreno:

[W3,S] = perform_fast_marching(W, IJK, options);


W3(isinf(W3))=0;
W3(isnan(W3))=0;
W3=smooth3(W3);
W3=rescale(W3,0,1);  % Aqui hay que cambiar el rescalado segun la
                     % fuerza que deseemos. Al final corresponde a
                     % la corriente ascendente debida al incendio.

W3=min(W,W3);

 % % SE PREPARAN LOS INPUTS PARA LA SEGUNDA LLAMADA A FAST MARCHING



start_points=[];
j=1;
for i=1:a
    for k=1:h
        if k>round(Z2(i,j))
            start_points=[start_points,[i;j;k]];
        end
    end
end
```

```
% Se llama a Fast Marching para producir expansiones de onda en cada
% punto de la cara Este de W3 sobre la matriz W3 que ya cuenta con la
% expansión de onda de cada punto de la superficie. De tal forma que el
% output de esta segunda llamada será una matriz de distancias que
% cuenta con 2 expansiones de onda.

[D,S] = perform_fast_marching(W3, start_points, options);
D(isinf(D))=0;
D(isnan(D))=0;
D=smooth3(D);


[DX,DY,DZ]=gradient(D);
DX(isnan(DX))=0;DY(isnan(DY))=0;DZ(isnan(DZ))=0;
[X,Y] = meshgrid(1:300,1:300);
Z3=Z2(1:300,1:300);
DX2=zeros(300,300); DY2=zeros(300,300); DZ2=zeros(300,300);

for i=1:300;
    for j=1:300;
        DX2(i,j)=DX(i,j,round(Z3(i,j))+4);
        DY2(i,j)=DY(i,j,round(Z3(i,j))+4);
        DZ2(i,j)=DZ(i,j,round(Z3(i,j))+4);
    end
end

% En esta línea ya se han rellenado las matrices que contienen la
% informacion sobre las distancias tanto para el eje X (DX2) como para
% el eje Y (DY2) y se procede a guardarlas para poder hacer uso de
% ellas en cada simulación directamente en lugar de tener que correr
% esta función y rellenarlas cada vez:

save('Che_EO_X.mat','DX6');
save('Che_EO_Y.mat','DY6');
```

C.8. Función `vientoORO`

```
function [vientoOrografia_X, vientoOrografia_Y]=vientoORO(alfa, cheoshe)

load Che_NS_X;    %  Carga la matriz DX1
load Che_NS_Y;    %  Carga la matriz DY1

load She_NS_X;    %  Carga la matriz DX2
load She_NS_Y;    %  Carga la matriz DY2
```

```
load Che_OE_X;   %  Carga la matriz DX3
load Che_OE_Y;   %  Carga la matriz DY3

load She_OE_X;   %  Carga la matriz DX4
load She_OE_Y;   %  Carga la matriz DY4

load Che_SN_X;   %  Carga la matriz DX5
load Che_SN_Y;   %  Carga la matriz DY5

load She_SN_X;   %  Carga la matriz DX6
load She_SN_Y;   %  Carga la matriz DY6

load Che_EO_X;   %  Carga la matriz DX7
load Che_EO_Y;   %  Carga la matriz DY7

load She_EO_X;   %  Carga la matriz DX8
load She_EO_Y;   %  Carga la matriz DY8



%  BUCLE CHEROKEE

if ((0<=alfa) && (alfa<=90) && (cheoshe==1))


viEO_X=DX7;
viEO_Y=DX7;

viSN_X=DX5;
viSN_Y=DY5;

vientoOrografia_X = sin(alfa) * viEO_X + cos(alfa) * viSN_X;
vientoOrografia_Y = sin(alfa) * viEO_Y + cos(alfa) * viSN_Y;

elseif ((90<alfa) && (alfa<=180) && (cheoshe==1))

viEO_X=DX7;
viEO_Y=DY7;

viNS_X=DX1;
viNS_Y=DY1;

vientoOrografia_X = sin(180-alfa) * viEO_X + cos(180-alfa) * viNS_X;
vientoOrografia_Y = sin(180-alfa) * viEO_Y + cos(180-alfa) * viNS_Y;

elseif ((180<alfa) && (alfa<=270) && (cheoshe==1))

viOE_X=DX3;
viOE_Y=DY3;

viNS_X=DX1;
viNS_Y=DY1;

vientoOrografia_X = sin(alfa-180) * viOE_X + cos(alfa-180) * viNS_X;
```

```
vientoOrografia_Y = sin(alfa-180) * viOE_Y + cos(alfa-180) * viNS_Y;

elseif ((270<alfa) && (alfa<360) && (cheoshe==1))

viOE_X=DX3;
viOE_Y=DY3;

viSN_X=DX5;
viSN_Y=DY5;

vientoOrografia_X = sin(360-alfa) * viOE_X + cos(360-alfa) * viSN_X;
vientoOrografia_Y = sin(360-alfa) * viOE_Y + cos(360-alfa) * viSN_Y;

elseif ((alfa==360))

dim1=300;
vientoOrografia_X = ones(dim1);
vientoOrografia_Y = ones(dim1);

end


% BUCLE SHERMAN


if ((0<=alfa) && (alfa<=90) && (cheoshe==0))


viEO_X=DX8;
viEO_Y=DX8;

viSN_X=DX6;
viSN_Y=DY6;

vientoOrografia_X = sin(alfa) * viEO_X + cos(alfa) * viSN_X;
vientoOrografia_Y = sin(alfa) * viEO_Y + cos(alfa) * viSN_Y;

elseif ((90<alfa) && (alfa<=180) && (cheoshe==0))

viEO_X=DX8;
viEO_Y=DY8;

viNS_X=DX2;
viNS_Y=DY2;

vientoOrografia_X = sin(180-alfa) * viEO_X + cos(180-alfa) * viNS_X;
vientoOrografia_Y = sin(180-alfa) * viEO_Y + cos(180-alfa) * viNS_Y;

elseif ((180<alfa) && (alfa<=270) && (cheoshe==0))

viOE_X=DX4;
viOE_Y=DY4;

viNS_X=DX2;
```

```
viNS_Y=DY2;

vientoOrografia_X = sin(alfa-180) * viOE_X + cos(alfa-180) * viNS_X;
vientoOrografia_Y = sin(alfa-180) * viOE_Y + cos(alfa-180) * viNS_Y;

elseif ((270<alfa) && (alfa<360) && (cheoshe==0))

viOE_X=DX4;
viOE_Y=DY4;

viSN_X=DX6;
viSN_Y=DY6;

vientoOrografia_X = sin(360-alfa) * viOE_X + cos(360-alfa) * viSN_X;
vientoOrografia_Y = sin(360-alfa) * viOE_Y + cos(360-alfa) * viSN_Y;

elseif ((alfa==360))

dim1=300;
vientoOrografia_X = ones(dim1);
vientoOrografia_Y = ones(dim1);

end

if (vientoOrografia_X(1,1) ~= 1)

vientoOrografia_X=rescale(vientoOrografia_X,0,1);
vientoOrografia_Y=rescale(vientoOrografia_Y,0,1);

end

end
```

C.9. Función ejecuta_FM_prueba_R

```
function ejecuta_FM_prueba_R(elevacion,velocidad,tiempo, vientoX, vientoY, sX,
sY, eX, eY, limTiempo, cheoshe)

%Ejecucuta el algoritmo FM y adicionalmente representa grï¿½ficamente la
propagaciï¿½n del
%fuego sobre el mapa de elevaciï¿½n que se le proporcione

%elevacion es la matriz con los datos de elevaciï¿½n del mapa en cuestiï¿½n
%velocidad es la matriz de coste o velocidad para introducir en el
%algortimo FM, tendrï¿½ la misma orientaciï¿½n y dimensiones que elevacion
%tiempo es el valor de tiempo del slider
%vientoX y vientoY son los valores del viento en las direcciones X e Y
%respectivamente de los sliders
%sX, sY, eX y eY puntos iniciales y finales respectivamente.
%limTiempo es el rango mï¿½ximo que cubre el slider tiempo, que deberï¿½a
```

```
%cambiar dentro del GUI en funciï¿½n del valor de los starting y ending
%points

hold off

[dim1,dim2]=size(elevacion);%dimensiones de la matriz, necesarias para meter
en el algoritmo

%Rango de alturas con el que se trabaja en la matriz elevacion

maxAltura=max(max(elevacion));
minAltura=min(min(elevacion));
difAltura=maxAltura-minAltura;

%Orientar la matriz para que se pueda ejecutar el algoritmo

velocidad=flip_vertical(velocidad);
velocidad = velocidad';

%Wo es la matriz de elevaciï¿½n rotada para poder ejecutar el algoritmo.

Wo1=rescale(elevacion,0.1,0.9);
Wo1=flip_vertical(Wo1);
Wo1 = Wo1';
Wo=1-Wo1;

[DX,DY] = gradient(Wo);%efecto de la montaï¿½a viene dado por el gradiente de
del mapa de alturas original, no del campo de velocidades.
%Por decidir si tiene efecto relevante sobre la propagaciï¿½n del fuego.

alfa=angulo(vientoX, vientoY);
[viorox, vioroy]=vientoORO(alfa, cheoshe);

%Campos vectoriales en direcciï¿½n X e Y
K=0.1;
FX =-0.5*vientoX*viorox(dim1)+DX;
minFX=min(min(FX));maxFX=max(max(FX));

FX=rescale(FX,minFX,maxFX);

FY=0.5*vientoY*vioroy(dim2)+DY;
minFY=min(min(FY));maxFY=max(max(FY));
FY=rescale(FY,minFY,maxFY);



N=ones(dim1,dim2);

%La imagen sale invertida, y para que la coordenada y que se introduce
%coincida con la real en al grï¿½fico

sY1=dim2-sY;
eY1=dim2-eY;
```

```
%Se inicializa el punto inicial o foco y se desiigna el punto final

start_points = [sX;sY1];
end_points = [eX;eY1];

% FMM as usual (options)

options.nb_iter_max = inf;
options.end_points = end_points;

%velocidad=rescale(velocidad,0.1,0.9);

[D,S2] = fast_marching_vectorial_2d(velocidad, start_points, options, FX,FY,
N); %ejecutamos algoritmo FM

d=flip_vertical(D');%"Desrotamos" la matriz D

nNiveles=200;%Nï¿½mero de niveles del contorno

C=contour(d,nNiveles); %Matriz con la informaciï¿½n de los contornos de d.
Elegimos cuantos niveles queremos

%Usaremos la propia matriz C para guardar los valores Z del contorno
%(aquellos correspondientes a W

longitudC=length(C);

for i=1:longitudC

    if C(2,i)<=dim1 %Solucionar problemas con los indicadores de nivel.
        %Sï¿½, hallamos las alturas de los indicadores de nivel, pero
computacionalmente cuesta menos que hacerlos desaparecer

        C(3,i)=elevacion(ceil(C(2,i)), ceil(C(1,i)));%Por alguna extraï¿½a
razï¿½n X e Y tienen que estar invertidos. Ceil redondea hacia arriba.

    end
end

%Sacado y ploteado de los niveles sobre la matriz de elevaciï¿½n

mesh(elevacion); hold on %Ploteamos primero la montaï¿½a en 3D

%Le ponemos el colormap para la montaï¿½a

zlimits = [min(elevacion(:)) max(elevacion(:))];
demcmap(zlimits);

%Primer bucle para sacar los niveles que verdaderamente hay (por alguna
%razï¿½n en realidad hay mï¿½s de los establecidos, son pequeï¿½os circulitos)

ant=1;
color=0;
```

```
final=longitudC*tiempo/limTiempo;

while ant<final

color=color+1;
num=C(2,ant);
ant=ant+1+num;

end

%La variable color tiene el nï¿½mero de verdaderos niveles que hay

j=jet(color); %Queremos que la propagaciï¿½n de la onda se haga acorde con los
colores del colormap jet.
%Sacamos un vector con la informaciï¿½n, con tantas celdas como verdaderos
niveles

%Usamos el loop anterior pero esta vez ploteando cada uno de los niveles

ant=1;
color=0;

while ant<final

color=color+1;
num=C(2,ant);
cX=C(1, (ant+1):(num+ant));
cY=C(2, (ant+1):(num+ant));
cZ=C(3, (ant+1):(num+ant))+0.02*difAltura*ones(1,length(cY));%Sumamos una
pequeï¿½a altura para mejor visualizaciï¿½n

plot3(cX,cY,cZ,'Color', j(color,:));

ant=ant+1+num;

end

%Representación de los puntos de inicio y final

scatter3(sX,sY,elevacion(sY,sX)+0.025*difAltura,'o','MarkerFaceColor','k');
%scatter3(eX,eY,elevacion(eY,eX)+0.025*difAltura,'o','MarkerFaceColor','w');
```

C.10. GUI

```
function varargout = trasteando(varargin)
% TRASTEANDO MATLAB code for trasteando.fig
%      TRASTEANDO, by itself, creates a new TRASTEANDO or raises the existing
%      singleton*.
```

```
%
%      H = TRASTEANDO returns the handle to a new TRASTEANDO or the handle to
%      the existing singleton*.
%
%      TRASTEANDO('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in TRASTEANDO.M with the given input arguments.
%
%      TRASTEANDO('Property','Value',...) creates a new TRASTEANDO or raises
the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before trasteando_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to trasteando_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help trasteando

% Last Modified by GUIDE v2.5 19-May-2017 13:02:15

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @trasteando_OpeningFcn, ...
                   'gui_OutputFcn',  @trasteando_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before trasteando is made visible.
function trasteando_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to trasteando (see VARARGIN)



% Choose default command line output for trasteando
```

```
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes trasteando wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = trasteando_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on slider movement.
function sliderVientoEO_Callback(hObject, eventdata, handles)
% hObject    handle to sliderVientoEO (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
slider

%Mï¿½ximo y mï¿½nimo del slider

set(hObject,'Max',5,'Min',-5);

%Redondeo del valor del slider y display como texto

valor=round(get(handles.sliderVientoEO,'Value'));

set(handles.numero_vientoEO,'String',valor);




% --- Executes during object creation, after setting all properties.
function sliderVientoEO_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sliderVientoEO (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
```

```
end


% --- Executes on selection change in popupmenu2.
function popupmenu2_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu2 contents
as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu2


% --- Executes during object creation, after setting all properties.
function popupmenu2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on slider movement.
function sliderTiempo_Callback(hObject, eventdata, handles)
% hObject    handle to sliderTiempo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
slider


%Se cogen los parï¿½metros necesarios de la estructura handles

vientoX=round(get(handles.sliderVientoEO,'Value'));
vientoY=round(get(handles.sliderVientoNS,'Value'));

velocidad=handles.velocidad;

sX=str2double(get(handles.sX,'String'));
sY=str2double(get(handles.sY,'String'));
eX=str2double(get(handles.eX,'String'));
eY=str2double(get(handles.eY,'String'));

%Cï¿½lculo del tiempo total mediante la funciï¿½n calculateLimTiempo

limTiempo=calculateLimTiempo(velocidad, sX,sY,eX,eY, vientoX, vientoY);
```

```
%Redondeo y paso a minutos

limTiempo=round(limTiempo*60);

%Mï¿½ximo y mï¿½nimo del slider

set(hObject,'Max',limTiempo,'Min',0);%Establecer dicho valor en el slider para
esos datos

%Redondeo del valor del slider y display como texto

valor=round(get(handles.sliderTiempo,'Value'));
set(handles.numero_tiempo,'String',valor);









% --- Executes during object creation, after setting all properties.
function sliderTiempo_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sliderTiempo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu1 contents
as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu1


% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in popupmenu3.
function popupmenu3_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu3 contents
as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu3


% --- Executes during object creation, after setting all properties.
function popupmenu3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on slider movement.
function sliderVientoNS_Callback(hObject, eventdata, handles)
% hObject    handle to sliderVientoNS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
slider

%Mï¿½ximo y mï¿½nimo del slider

set(hObject,'Max',5,'Min',-5);

%Redondeo del valor del slider y display como texto

valor=round(get(handles.sliderVientoNS,'Value'));
set(handles.numero_vientoNS,'String',valor);



% --- Executes during object creation, after setting all properties.
function sliderVientoNS_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to sliderVientoNS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



function sX_Callback(hObject, eventdata, handles)
% hObject    handle to sX (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of sX as text
%        str2double(get(hObject,'String')) returns contents of sX as a double

matrix=handles.elevacion;

[dim1, dim2]=size(matrix);

sX=str2double(get(hObject,'String'));

%Se introducen restricciones de los valores que se pueden meter para que
%esten comprendidos entre 1 y dim1

if((sX>dim1)||(sX<1))
errordlg('Valor No Admitido','Curso_GUIDE');
end




% --- Executes during object creation, after setting all properties.
function sX_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sX (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function sY_Callback(hObject, eventdata, handles)
% hObject    handle to sY (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of sY as text
%        str2double(get(hObject,'String')) returns contents of sY as a double

matrix=handles.elevacion;

[dim1, dim2]=size(matrix);

sY=str2double(get(hObject,'String'));

%Se introducen restricciones de los valores que se pueden meter para que
%esten comprendidos entre 1 y dim2
if((sY>dim2)||(sY<1))
errordlg('Valor No Admitido','Curso_GUIDE');
end

% --- Executes during object creation, after setting all properties.
function sY_CreateFcn(hObject, eventdata, handles)
% hObject    handle to sY (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function eX_Callback(hObject, eventdata, handles)
% hObject    handle to eX (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of eX as text
%        str2double(get(hObject,'String')) returns contents of eX as a double

matrix=handles.elevacion;

[dim1, dim2]=size(matrix);

eX=str2double(get(hObject,'String'));
%Se introducen restricciones de los valores que se pueden meter para que
%esten comprendidos entre 1 y dim1
if((eX>dim1)||(eX<1))
%handles.inix=100;
errordlg('Valor No Admitido','Curso_GUIDE');
end

% --- Executes during object creation, after setting all properties.
```

```
function eX_CreateFcn(hObject, eventdata, handles)
% hObject    handle to eX (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function eY_Callback(hObject, eventdata, handles)
% hObject    handle to eY (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of eY as text
%        str2double(get(hObject,'String')) returns contents of eY as a double

matrix=handles.elevacion;

[dim1, dim2]=size(matrix);

eY=str2double(get(hObject,'String'));

%Se introducen restricciones de los valores que se pueden meter para que
%esten comprendidos entre 1 y dim2

if((eY>dim2)||(eY<1))
errordlg('Valor No Admitido','Curso_GUIDE');
end

% --- Executes during object creation, after setting all properties.
function eY_CreateFcn(hObject, eventdata, handles)
% hObject    handle to eY (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in simular.
function simular_Callback(hObject, eventdata, handles)
% hObject    handle to simular (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
%Se cogen los parï¿½metros necesarios de la estructura handles

elevacion=handles.elevacion;
velocidad=handles.velocidad;
cheoshe=handles.cheoshe;

tiempo=round(get(handles.sliderTiempo,'Value'));

vientoX=round(get(handles.sliderVientoEO,'Value'));
vientoY=round(get(handles.sliderVientoNS,'Value'));
limTiempo=get(handles.sliderTiempo,'Max');

sX=str2double(get(handles.sX,'String'));
sY=str2double(get(handles.sY,'String'));
eX=str2double(get(handles.eX,'String'));
eY=str2double(get(handles.eY,'String'));

%Se preparan los ejes del GUI
axes(handles.axes1);

%Uso de la funciï¿½n ejecuta_FM
ejecuta_FM_prueba_R(elevacion, velocidad, tiempo, vientoX, vientoY, sX, sY,
eX, eY, limTiempo, cheoshe)



% --- Executes on button press in buscarMapa.
function buscarMapa_Callback(hObject, eventdata, handles)
% hObject    handle to buscarMapa (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%el archivo .mat que se coja ha de contener 2 matrices de las mismas
%dimensiones llamadas elevacion y velocidad

filename=uigetfile('*.mat','Elegir archivo .mat');%Menï¿½ para conseguir las
matrices
S = load(filename);%Estructura que las carga en el workspace actual

%Se pasa a la estructura handles para que pueda ser utilizada en otras
%partes del GUI
handles.elevacion=S.elevacion;
handles.velocidad=S.velocidad;
handles.cheoshe=S.CheOshe;

guidata(hObject,handles);


% --- Executes on button press in sinPropagacion.
function sinPropagacion_Callback(hObject, eventdata, handles)
% hObject    handle to sinPropagacion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

%Se preparan los ejes del GUI
axes(handles.axes1);

hold off

%Representaciï¿½n de la matriz elevacion con el colormap de montaï¿½a

mesh(handles.elevacion);

zlimits = [min(handles.elevacion(:)) max(handles.elevacion(:))];
demcmap(zlimits);


% --- Executes on button press in combustion.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to combustion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

figure

velocidad=flip_vertical(handles.velocidad);%Giro de la imagen para que
coincida con la orientaciï¿½n de la matriz de elevacion

imagesc(velocidad);
```

REFERENCIAS

[1] H. Bruyninckx, “Preparing Europe for Climate Change: Coordination is key to reduce risks posed by extreme weather” European Environment Agency, Denmark, 2017.

[2] (2018, Aug.) WWF. [Online]. Available: https://wwf.panda.org/

[3] L. Hernández, A. Asiaín, G. Prudencio and E. S. Sivela, “Informe incendios 2018: El polvorín del noroeste” WWF/Adena, España, 2018.

[4] (2018, Aug.) Greenpeace. [Online]. Available: https://www.greenpeace.org/

[5] (2018, Aug.) El País. [Online]. Available: https://elpais.com/

[6] A. Burgueño, “Modelización de un incendio forestal mediante fast marching,” Master´s Thesis, Universidad Carlos III de Madrid, 2015.

[7] C. N. Martín, “Modelización de un incendio forestal con Fast Marching,” Bachelor´s Thesis, Universidad Carlos III de Madrid, 2017.

[8] R. Sun, S. K. Krueger, M. A. Jenkins, M. A. Zulauf, and J. J. Charney, "The importance of fire atmosphere coupling and boundary-layer turbulence to wildfire spread," *International Journal of Wildland Fire 18(1) 50-60,* 2009.

[9] J. S. Gould, N. Cheney, L. McCaw and S. Cheney, "Effects of head fire shape and size on forest fire rate of spread", *Conference Proceedings of 3rd International Wildland Fire (pp. 3-6),* 2003.

[10] M. Guijarro, J. Madrigal, C. Díez, and J. S. Martín, "Caracterización de la propagación del fuego en matorral de carqueixa mediante quemas en túnel de viento," *Congresos-CARGA FINAL,* 2005.

[11] P. H. Kourtz and O'Regan, "A model a small forest fire to simulate burned and burning areas for use in a detection model. forest science," Forest *Science, Volume 17, Number 2, 1 June 1971, pp. 163-169(7),* 1971.

[12] V. Mallet, D. Keyes, and F. Fendell, "Modeling wildland fire propagation with level set methods," *Computers and Mathematics with Applications 57 1089-1101,* 2009.

[13] J. Hilton, C. Miller, A. Sullivan, and C. Rucinski, "Efects of spatial and temporal variation in environmental conditions on simulation of wildfire spread," *Environmental Modelling & Software 67 118-127,* 2015.

[14] R. C. Rothermel, "A mathematical model for predicting fire spread in wildland fuels." 1972.

[15] (2018, Aug.) Firers. [Online]. Available: http://www.fire-rs.com/es/

[16] C. Tymstra, R. Bryce, B. Wotton, S. Taylor, and O. Armitage, "Development and structure of prometheus: the canadian wildland fire growth simulation model," Canadian Forest Service, Tech. Rep., 2010.

[17] V. López, "¿En qué consiste el principio de Huygens?", *VARINIA.ES,* 19 December 2010. [Online]. Available: http://varinia.es/blog/2010/12/19/

[18] *Burn-CP3 Version 4.7 User's manual, 2017.*

[19] D. Morvan, J. Dupuy, E. Rigolot, and J. Valette, "Firestar: A physically based model to study wildfire behaviour," *Forest Ecology and Management 234S S114,* 2006.

[20] M. A. Finney, "Farsite: Fire area simulator-model development and evaluation." 2004.

[21] (2018, Aug.) Fireorg. [Online]. Available: http://www.fire.org/

[22] J. V. Gómez, “Fast marching methods in path and motion planning: Improvements and high-level applications," Ph.D. dissertation, Universidad Carlos III Madrid, 2015.

[23] S. Osher and J. A. Sethian, “Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulations.” *Journal of computational physics, 79(1), 12-49.*, 1988.

[24] (2018, Aug.) Numerical tours. [Online]. Available: http://www.numerical-tours.com/

[25] (2018, Aug.) Mathworks. [Online]. Available: https://es.mathworks.com/

[26] Unknown Author “Efecto Föhn”, *Wikipedia*, 4 October 2017. [Online]. Available: https://es.wikipedia.org/wiki/Efecto_F